PROGRAMMEZ.COM

PROGRAMMEZ!

le magazine du développeur

Mensuel n°189 - Octobre 2015

python™ Un langage à connaître absolument !

node JS® de A à Z (re)découvrez JavaScript côté serveur

BD Connaissez-vous **Les Geeks ?**

Le Godmode de Windows 10

Mean.io Votre nouveau framework Web

Les nouveautés d' Unity 5.x

Vous rêvez de Silicon Valley ? Témoignages & conseils

Level 400 Les templates **C++**

Java 8 Maîtriser l'API Date and Time



Printed in EU - Imprimé en UE - BELGIQUE 6,45 € SUISSE 12 FS - LUXEMBOURG 6,45 € - DOM Surf 6,90 € Canada 8,95 $ CAN - TOM 940 XPF - MAROC 50 DH


iKoula
HÉBERGEUR CLOUD
TROPHÉES DUCLOUD
by EuroCloud
MEILLEUR SERVICE CLOUD
D'INFRASTRUCTURE 2015



Python qui peut...

Les développeurs sont-ils aussi conservateurs que cela sur les langages ? Sans doute un peu. Et pourtant, il y a des langages qui méritent d'être connus. L'un d'eux est Python. Ce langage n'est pas nouveau et il a largement fait ces preuves depuis 15 ans, en plus d'être beaucoup utilisé dans les outils d'infrastructures. Ce mois-ci, nous vous proposons une petite virée avec notre serpent favori : introduction au langage (les 2 branches, la syntaxe, les outils...) et comment utiliser Python avec les données.

Autre (re) découverte rien que pour vous : JavaScript, et plus précisément, Node.JS. Node.JS est, pour rappel, JavaScript côté serveur. Il devient ainsi un véritable langage côté serveur et non plus uniquement en frontal. Rappelons qu'il ne s'agit pas d'un framework, mais d'un véritable environnement « bas niveau » et des frameworks permettent justement d'éviter les opérations « bas niveau ».

Node.JS méritait bien un dossier complet, en 2 parties !

Level 400 : même pas peur !

Dans ce numéro, nous avons aussi plusieurs articles, disons, un peu hardcore. Les templates en C++ tu connais ? Non, là, tu vas apprendre à les aimer (ou pas). Notre Jedi C++ a décidé de vous entraîner, petits padawans.

Autre petit 400, rien que pour vous, gentils développeurs, le Deep Learning. Il s'agit d'un sous-domaine (ou d'un nouveau domaine, c'est selon) du Machine Learning. Il doit, à terme, faciliter l'Intelligence Artificielle. Pour faire plus léger, nous vous proposons une plongée dans le framework Mean.IO et dans les nouveaux modules de date et heure de Java 8, qui simplifient enfin le travail du développeur !

Ouf, il n'y a pas que du Level 400 !

Tu es tenté(e) par la Silicon Valley ? La rédaction a mené l'enquête sur place pour voir comment cela se passe réellement et vous verrez que la réalité est souvent moins glamour que ce que l'on voit dans les médias français. Il y a des opportunités, mais il faut en vouloir, et sortir du lot. La concurrence est rude.

Nous vous proposons aussi de découvrir une BD sympathique et qui nous a immédiatement plu : Les Geeks.

*C'est cadeau !**

Stressé par une démo ou une mise en production de son code ?



© Lucasfilm / Paramount

** fonctionne avec tout*

Le boss
ftonic@programmez.com

à lire dans le prochain numéro n°190 en kiosque le 31 Octobre 2015

Office 365
Découvrez les dernières API et opportunités d'Office pour le développeur

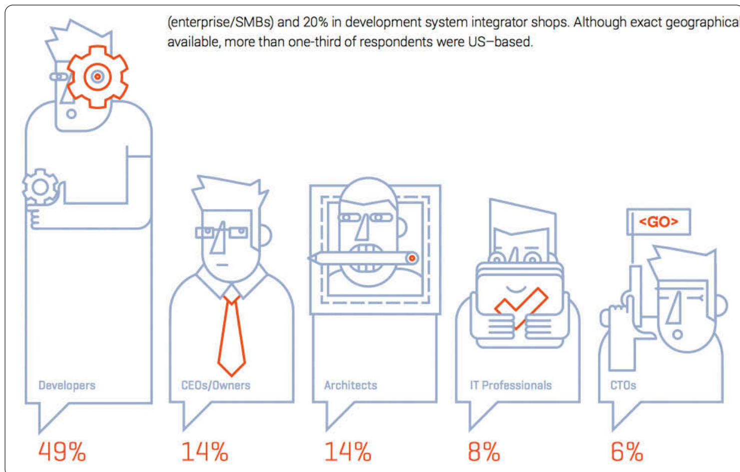
Cobol
Un langage pas si mort !

App Store : qui veut gagner des millions ?
App Store, on en parle tout le temps. Mais c'est quoi exactement ? Plongez dans les différents stores, les bonnes pratiques, les retours d'expérience, qui paie le mieux les développeurs ?

Les défis du développement pour le développeur

L'éditeur Telerik (filiale de Progress) a dévoilé fin août son rapport « state of mobility survey 2015 », grande enquête menée sur 3000 développeurs à travers le monde. Une des surprises (?) est la première donnée de l'étude : le déploiement et la disponibilité des apps mobiles sont ralentis par le manque de ressources ! Et le constat est même sévère : 57 % des développeurs disent avoir en développement qu'une seule app et qu'ils sortent une app mobile par an...

Plusieurs raisons sont données pour expliquer cette situation : manque de temps, trouver le bon développeur, les changements de technologies ou de pratiques de développement, le problème du multiplateforme, les contraintes budgétaires...



Sans surprise, pour les développeurs, il faut demander des apps attractives ! L'amélioration de l'efficacité opérationnelle est l'une des raisons les plus couramment citées pour la création d'apps mobiles. Parmi les autres raisons évoquées figurent la génération de nouvelles sources de revenus (39 %), l'augmentation de la productivité des employés (38 %), l'amélioration du service client (35 %) et un engagement client plus pertinent (34 %). Pour atteindre ces différents objectifs, l'expérience utilisateur (UX) est l'élément le plus important pour les développeurs d'applications mobiles — que ces applis soient pour un public externe ou interne à l'entreprise.

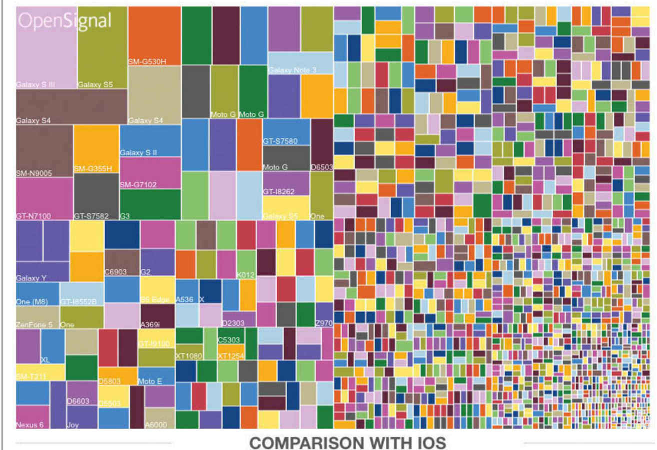
Les développeurs interrogés sont 44 % à citer l'expérience client comme élément principal des applications qu'ils créent, avant la facilité de maintenance (24 %), les performances (15 %) et la sécurité (11 %).

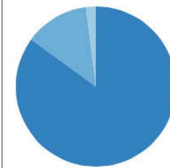
Pas de natif et du multiplateforme

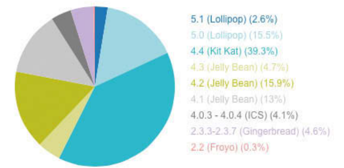
Pour les entreprises et les éditeurs, l'approche hybride est une nécessité, pour un tiers des sondés, mais le natif arrive à 25 %. Bref les avis sont très partagés et le natif résiste toujours. Android et iOS sont toujours largement devant (76 % et 63 %) chez les développeurs, mais Windows Phone arrive tout de même à 40 %. Cependant, le multiplateforme est une nécessité pour 36 % ce qui est beaucoup, mais finalement pas écrasant.

Enfin, plusieurs challenges sont déjà aux portes du code pour les développeurs. L'étude met en avant : des développements au-delà du mobile (web, desktop), les objets connectés et le wearable (mais peu représentatif pour le moment).

TU AIMES LA FRAGMENTATION ?







Opensignal.com sort chaque année une étude sur la fragmentation Android. C'est toujours instructif et impressionnant. La plate-forme Android compte plus de 18 000 terminaux différents et 1 294 marques. Samsung domine largement, mais on constate tout de même que le constructeur introduit une large fragmentation avec ses nombreuses gammes ! Mais finalement, le constructeur domine dans de nombreux pays ce qui « facilite » le travail du développeur, mais il doit tout de même supporter des formats d'écrans de plus en plus en nombreux et actuellement, Android 4.4 domine toujours le parc Android alors que la ligne 5.0 plafonne à 18 % ! Côté iOS, c'est très simple : 85 % des terminaux sous iOS 8.x, 2 % sous 7 et 2 % avec une version plus ancienne...

À lire d'urgence : http://opensignal.com/reports/2015/08/android-fragmentation/

Pas de chiffres officiels, mais c'est pas grave !

Ah la magie des chiffres et des stats. Apple ne fournit pas de chiffres officiels sur les ventes de sa montre, mais cela n'empêche pas de sortir des chiffres précis. Selon Tractica, l'Apple Watch occupera 68 % du marché à la fin de l'année. Le marché global de la montre intelligente sera d'environ 24 millions (d'unités), Apple pourrait en fournir presque 17 millions... Samsung sera 2e avec 12 %. Plus globalement, le marché du wearable est en pleine explosion même s'il reste encore petit en volume. Selon IDC, ce marché pèse 18 millions d'unités. Fitbit est 1er avec 4,4 millions d'objets vendus, suivi par Apple. Samsung est très loin derrière.

Fabriquant	millions d'unités 2015	% marché (2e trimestre 2015)	millions d'unités (2e trimestre 2014)	% marché (2e trimestre 2014)
1. Fitbit	4.4	24.3 %	1.7	30.4 %
2. Apple	3.6	19.9 %	0	0.0 %
3. Xiaomi	3.1	17.1 %	0	0.0 %
4. Garmin	0.7	3.9 %	0.5	8.9 %
5. Samsung	0.6	3.3 %	0.8	14.3 %
Autres	5.7	31.5 %	2.6	46.4 %
Total	18.1	100.0 %	5.6	100.0 %

Source : IDC, 27 août 2015

Quoi qu'il en soit, Apple, même si on peut juger les premières ventes faibles, a réussi à dynamiter la montre connectée et le wearable en à peine 6 mois. Mais il est clair que la Pomme devra améliorer les fonctionnalités (notamment avec le prometteur watchOS 2) et le matériel.

Raspberry Pi 2 : le transformer en PC !

Vous cherchez une machine pas chère, un peu fun, sous Linux ? Avez-vous pensé au Raspberry Pi 2 sorti au printemps dernier ? La carte seule coûte environ 35 € et est un mini-PC complet. Reste à connecter tous les périphériques et à lui trouver un joli boîtier...



François Tonic





Vous cherchez un barebone ou « simple » PC qui fera les tâches élémentaires (bureautique, Internet, un peu de programmation) ? Pas besoin d'une grosse machine, un Raspberry Pi 2 peut largement suffire. Brut, ce n'est pas très design, mais vous pouvez trouver des kits complets ou des boîtiers pour y installer la carte et y connecter, proprement, clavier, souris, écran, réseau...

Brut de fonderie

Le reproche que l'on peut faire au Pi 2 est de venir sans rien. À vous d'avoir une carte SD et d'installer le système et les outils nécessaires, ou d'acheter une carte préinstallée. Ce n'est pas très compliqué, mais il faut un PC pour le faire; l'avantage est néanmoins d'être très souple et de s'adapter à vos besoins. Côté système, plusieurs distributions Linux sont disponibles, la plus classique est la Raspbian. La Pi 2 a donné un sérieux coup de fouet au système. Vous pouvez installer Windows 10 IoT mais il ne s'agit pas d'un Windows complet avec interface. Cette édition se destine aux objets connectés et non à transformer votre Pi 2 en PC Windows.

Quelques idées de kits

Kano computer kit

Ce projet financé par crowdfunding est assez simple : proposer un véritable ordinateur à monter soi-même, comme un jeu de construction. Le kit standard propose un Pi 2, un haut-parleur, un clavier avec zone tactile intégrée, un module Wi-Fi, les câbles, une carte SD avec le système Kano OS. Ce kit se veut ludique et très rapide à monter. Reste ensuite à le brancher à un écran. Un petit boîtier permet d'intégrer la Pi 2. Le kit est proposé à 149,99 €. Le délai de livraison est au minimum d'une semaine. Site : http://www.kano.me



Pi-Top : le top du kit Pi ?

C'est sans doute le projet le plus ambitieux et le plus complet pour construire un PC avec une Pi. Ce projet a lui aussi bénéficié du crowdfunding, avec succès. Le principe est très simple : proposer un boîtier d'un portable complet avec clavier, écran, batterie, etc. L'écran affiche une taille de 13 pouces et une autonomie de 10 heures. Le boîtier se veut très modulaire et il est imprimé en 3D. Il intègre aussi son propre système, Pi-top OS, un dongle Wi-Fi. Il intègre aussi une carte électronique pour gérer l'alimentation, les différentes LED, la connexion avec la Pi 2. Deux modèles sont proposés en précommande : le Pi-top sans la Pi 2 (264,99 $) et le Pi-top complète (299,99 $).

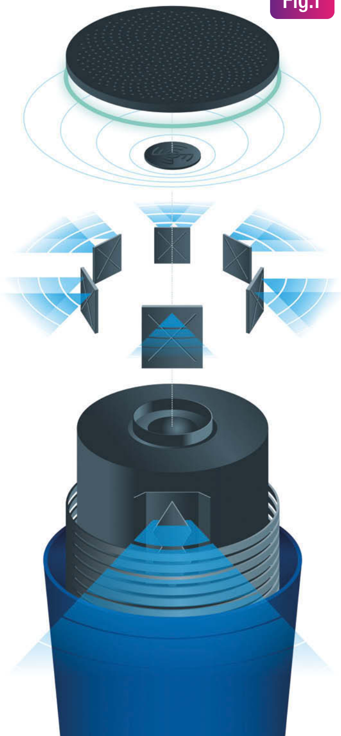
ZX Spectrum va-t-il renaître ?

Le vintage touche aussi l'informatique et les premiers ordinateurs... Un projet s'est animé pour faire renaître le vénérable ZX Spectrum qui a été un des ordinateurs les plus emblématiques du début des années 1980. Cette « nouvelle » machine s'appellera Sinclair ZX Spectrum Vega et il se base bien entendu sur le Spectrum et est orienté jeux. En quelques heures tous les exemplaires (dont les prix varient de 100 à 1000 £ pour le matériel et les pièces de collection) ont été épuisés ! Le projet a été entièrement financé. Les 1000 premiers exemplaires ont été expédiés début août. Site : http://www.zxvega.co.uk

Epiphan DVI2USB : capture audio depuis une source HDMI

Comment capturer simplement et rapidement une source audio depuis les ports HDMI ? Le construction Epiphan propose des solutions complètes. Les boitiers et cartes Epiphan AV.io HD, DVI2USB 3.0, DVI2PCIe Duo et tous les produits construits avec ces périphériques peuvent gérer les formats DVI, HDMI et les signaux VGA, mais également offrir un soutien pour la capture audio HDMI en utilisant un port compact DVI-I. Pour en savoir plus : epiphan-france.fr

Google lance son routeur Wi-Fi Fig.1


Fig.1

OnHub est le routeur Wi-Fi de Google. Il sera disponible en Amérique du Nord pour un prix de 199 $, aucune information pour une disponibilité ailleurs dans le monde. Ce routeur ressemble à un grand vase (le design est plutôt sympa) et comporte 13 antennes à 2,4 et 5 GHz, ce qui garantira une bonne connectivité. L'objet supporte les différentes déclinaisons du Wi-Fi 802.11 et pourra, dans une mise à jour, supporter d'autres protocoles. La gestion du routeur passe par des apps mobiles (Android et iOS). Selon Google, OnHub s'ajuste constamment pour proposer la meilleure connectivité et être à jour sans coupure réseau.
Site : https://on.google.com/hub/

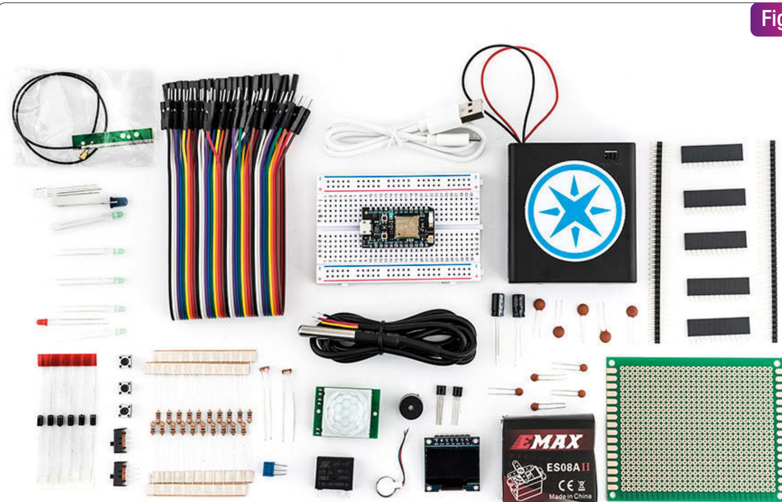
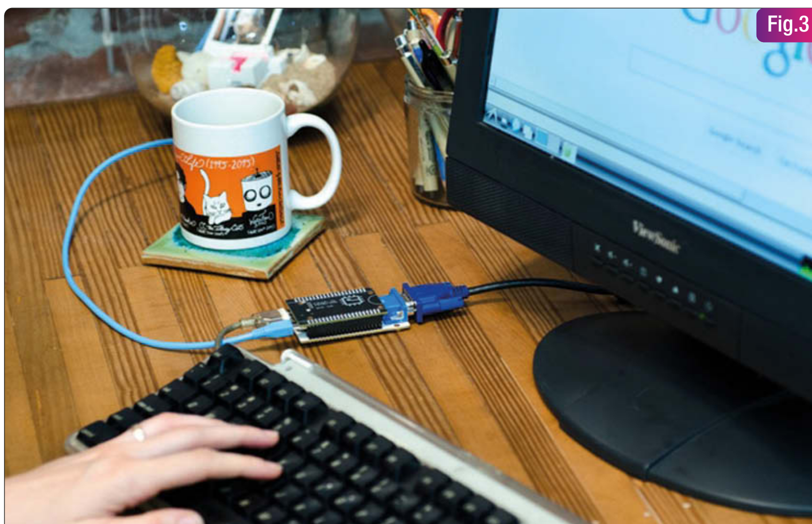


Fig.2


Fig.3

Disque WD Red 6 To

Le constructeur WD annonce la disponibilité des disques durs Red Pro en version 6 To pour les serveurs NAS. Ces modèles se veulent moins sensibles aux chocs. Une nouvelle version du firmware est aussi disponible. À partir de 290 € (modèle 5 To).

Les boards Photon disponibles (Particle)

Fig.2

Depuis plusieurs mois, nous étions des milliers à attendre la disponibilité d'une minuscule carte, la Photon (ex. : Spark). Ces cartes sont clairement dédiées aux Maker, mais surtout aux objets connectés. Toutes les versions intègrent un système réseau sans fil (Wi-Fi et/ou 2G/3G. Plusieurs versions sont disponibles : Photon, P0, P1, Spark Core et Electron. La photon est vendue à 19 $, l'Electron à 39. Des kits complets [avec composants, câble] démarrent à 29 $. Des shields ont été conçus sur mesure et un kit complet Grove [Seeed Studio] est même disponible [50 $]. Sparkfun, un constructeur très actif, propose aussi de nombreuses extensions, notamment un kit [115 $], très complet [et plus intéressant que le kit Grove].

C.H.I.P. arrive bientôt

Fig.3

Depuis le printemps, ce projet fait beaucoup de parler de lui. La promesse est simple : proposer une carte complète à l'instar du Raspberry Pi, mais avec deux avantages : une taille plus petite et surtout un prix très abordable, 9 $! Le projet a explosé son compteur Kickstarter en récoltant 2 000 000 $ [50 000 $ étaient demandés]. Pour beaucoup, c'est le Raspberry Pi killer. Mais avant de vouloir tuer la Pi 2, CHIP devra démontrer son potentiel : nous sommes certains qu'une forte communauté va rapidement apparaître avec un écosystème ouvert et dynamique. Il arrivera avec un système complet et des dizaines de logiciels [bureautique, développement, etc.].
La première version arrivera avec un processeur ARM 1 GHz, 512 Mo de mémoire vive, 4 Go de stockage, un module Wi-Fi/Bluetooth, USB, audio...
Des shields d'extensions sont déjà prévus pour étendre les fonctionnalités du CHIP. CHIP pourra facilement s'installer dans des bornes, un mini PC, vidéo, etc.
Les 10 000 premières cartes sont totalement épuisées et seront livrées entre décembre 2015 et janvier 2016. Une autre vague interviendra en février 2016 ! Une première édition [1000 exemplaires] sera livrée dès septembre [version « kernel hacker only »]. Cette version inclura une version alpha de CHIP.
Détail important : CHIP est open hardware !
Site : http://nextthing.co/

Google : le projet ARA repoussé à 2016

Le téléphone modulaire que l'on compose soi-même, projet ARA, est toujours en développement chez Google. Des démonstrations avaient été faites durant la Google I/O. Si le système semble désormais plus stable quand on installe à chaud des modules, un problème pratique repousse le projet. Quand le téléphone tombe par terre, tous les modules sont éjectés du socle. Les aimants ne sont pas assez puissants pour maintenir en place les modules. Pour le moment, les équipes cherchent une solution à ce problème...

1ER ÉVÉNEMENT EUROPÉEN
LIBRE & OPEN SOURCE



OPEN FOR INNOVATION

opensourcesummit.paris

#OSSPARIS15

PARIS OPEN SOURCE SUMMIT

18&19 NOVEMBRE

DOCK PULLMAN
Plaine Saint-Denis

SPONSORS PLATINUM



SPONSORS GOLD



PARTENAIRES

PARTENAIRES INSTITUTIONNELS



PARTENAIRES COMMUNAUTAIRES



SPONSORS SILVER



POUR TOUTE INFORMATION COMPLÉMENTAIRE :
Email : contact@opensourcesummit.paris – Tel : 01 41 18 60 52

un événement





Usurpation d'identité (User Impersonation)



Cédric Michel
Analyste développeur chez Trasys. Spécialiste .net Membre (MEET – Microsoft Extended Expert Team)
http://www.microsoft.com/belux/meet/



Cette procédure consiste à changer le contexte de sécurité dans lequel vous vous trouvez. Cela signifie que vous pourrez exécuter du code avec des droits qui correspondent à un autre utilisateur.
Cela peut être utilisé pour écrire des fichiers dans un répertoire particulier pour lequel seul un utilisateur a accès.

Exemple d'un cas concret :

Pour une application Web (MVC), il faut que celle-ci écrive dans un répertoire se trouvant sur un autre serveur Web. Cependant pour des règles de sécurité propres au client, il est impossible d'attribuer un utilisateur de l'Active Directory pour l'exécution de l'application pool. Afin de contourner cette règle de sécurité, l'utilisation de l'impersonation peut être utile. Grâce à cela, en créant un utilisateur sur l'Active Directory et en utilisant l'impersonation nous pourrons écrire des fichiers avec cet utilisateur. Voyons en détail la mise en place de l'impersonation. Pour cela, il faut appeler une fonction qui réside dans la DLL advapi32.dll. Il s'agit de la fonction LogonUser.

Syntax du code non managé en C++

```
BOOL LogonUser(
  _In_     LPTSTR lpszUsername,
  _In_opt_ LPTSTR lpszDomain,
  _In_opt_ LPTSTR lpszPassword,
  _In_     DWORD  dwLogonType,
  _In_     DWORD  dwLogonProvider,
  _Out_    PHANDLE phToken
);
```

Vous pouvez retrouver tous les détails de cette méthode ici :
https://msdn.microsoft.com/en-us/library/windows/desktop/aa378184(v=vs.85).aspx
Comme il s'agit de code non managé, il faut utiliser l'attribut DLLImport. Cette utilisation permet de faire appel à la fonction qui réside dans la DLL et de lancer son traitement.

```
[DllImport("advapi32.dll")]
 private static extern bool LogonUser(String lpszUserName,
    String lpszDomain,
    String lpszPassword,
    int dwLogonType,
    int dwLogonProvider,
    ref IntPtr phToken);
```

Les trois premiers paramètres sont utiles pour l'authentification de l'utilisateur. Il s'agit simplement du nom de l'utilisateur ainsi que du domaine et évidemment de son mot de passe. Pour les paramètres suivants, vous trouverez les valeurs possibles pour le logon type (dwLogonType) et le provider (dwLogonProvider), dans le fichier WinBase.h. Ce fichier se trouve dans le répertoire suivant : C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1A\Include
Tableau des valeurs possibles pour le paramètre dwLogonProvider

```
//
// Logon Support APIs
//

#define LOGON32_LOGON_INTERACTIVE       2
#define LOGON32_LOGON_NETWORK           3
#define LOGON32_LOGON_BATCH             4
#define LOGON32_LOGON_SERVICE           5
#define LOGON32_LOGON_UNLOCK            7
#if(_WIN32_WINNT >= 0x0500)
#define LOGON32_LOGON_NETWORK_CLEARTEXT 8
#define LOGON32_LOGON_NEW_CREDENTIALS   9
#endif // (_WIN32_WINNT >= 0x0500)
```

Afin d'avoir une utilisation interactive avec le système, l'utilisation du type « LOGON32_LOGON_INTERACTIVE » convient. Donc pour le paramètre dwLogonType qui est un type DWORD la valeur entière 2 sera utilisée.
Tableau des valeurs possibles pour le paramètre dwLogonType

```
#define LOGON32_PROVIDER_DEFAULT    0
#define LOGON32_PROVIDER_WINNT35    1
#if(_WIN32_WINNT >= 0x0400)
#define LOGON32_PROVIDER_WINNT40    2
#endif /* _WIN32_WINNT >= 0x0400 */
#if(_WIN32_WINNT >= 0x0500)
#define LOGON32_PROVIDER_WINNT50    3
#endif // (_WIN32_WINNT >= 0x0500)
#if(_WIN32_WINNT >= 0x0600)
#define LOGON32_PROVIDER_VIRTUAL    4
#endif // (_WIN32_WINNT >= 0x0600)
```

Nous utiliserons le provider par défaut (LOGON32_PROVIDER_DEFAULT) ce qui correspond à la valeur 0. Pour ce qui est du phToken, en réalité il s'agit d'un type PHANDLE. Ce type est un pointer d'Handle (descripteur) vers le token. En C# nous utiliserons donc le type IntPtr qui correspondra à ce pointeur. Comme il s'agit d'un paramètre « OUT » nous recevrons grâce au passage en référence le pointeur du token. Le pointeur du Token sera utilisé dans le constructeur de l'objet WindowsIdentity qui représente l'utilisateur. Grâce à cet objet nous pourrons dès lors faire l'emprunt de l'identité de l'utilisateur en question.

```
WindowsIdentity newIdentity = new WindowsIdentity(phToken);
_windowsImpersonationContext = newIdentity.Impersonate();
```

En retour de l'appel de la méthode Impersonate, vous recevrez un WindowsImpersonationContext. Cela correspond à l'utilisateur avant l'emprunt de son identité. Suite à l'ensemble de ces informations, j'ai créé un objet UserImpersonation. Pour en faire une utilisation correcte et facile cette classe implémente IDisposable. De cette manière, il vous sera facile de l'utiliser via un USING habituel. Cela aura également pour effet d'implémenter un dispose sur l'objet UserImpersonation. Dans ce dispose, la méthode Undo de l'objet WindowsImpersonationContext est appelée, ce qui a pour effet de rétablir le contexte de l'utilsateur. Le token qui vous a été renvoyé lors de la connexion de l'utilisateur doit impérativement être proprement clôturé. Pour se faire, un DLLImport de kernel32.dll doit être effectué et la méthode CloseHandle sera utilisée.

Syntaxe du code non managé en C++

```
BOOL WINAPI CloseHandle(
  _In_ HANDLE hObject
);
```

Vous pouvez retrouver tous les détails de cette méthode ici :
https://msdn.microsoft.com/en-us/library/windows/desktop/ms724211(v=vs.85).aspx
Cette méthode prend en paramètre le Handle du token.

```
[DllImport("kernel32.dll", CharSet = CharSet.Auto)]
public static extern bool CloseHandle(IntPtr handle);
```

Ce qui nous donne pour le Dipose de l'objet UserImpersonation ceci :

```
public void Dispose()
 {
   if (_windowsImpersonationContext != null)
     _windowsImpersonationContext.Undo();
   if (_tokenHandle != IntPtr.Zero)
     CloseHandle(_tokenHandle);
 }
```

Voici un exemple d'utilisation de l'objet.

```
using (UserImpersonation user = new UserImpersonation("login", "domain","password"))
{
    if (user.ImpersonateValidUser())
    {
       File.WriteAllText("Programmez.txt", "UserImpersonation");
    }
}
```

Vous pourrez constater que le propriétaire du fichier créé est bien l'utilisateur que vous passez en paramètre. Si vous désirez utiliser ce mécanisme, un package Nuget est à votre disposition.
https://www.nuget.org/packages/UserImpersonation/
Le code source est également disponible sur Github.
https://github.com/michelcedric/UserImpersonation
Code complet :

```
/// <summary>
/// Object to change the user authticated
/// </summary>
public class UserImpersonation : IDisposable
{
  /// <summary>
  /// Logon method (check athetification) from advapi32.dll
  /// </summary>
  /// <param name="lpszUserName"></param>
  /// <param name="lpszDomain"></param>
  /// <param name="lpszPassword"></param>
  /// <param name="dwLogonType"></param>
  /// <param name="dwLogonProvider"></param>
  /// <param name="phToken"></param>
  /// <returns></returns>
  [DllImport("advapi32.dll")]
  private static extern bool LogonUser(String lpszUserName,
    String lpszDomain,
    String lpszPassword,
    int dwLogonType,
    int dwLogonProvider,
    ref IntPtr phToken);

  /// <summary>
  /// Close
  /// </summary>
  /// <param name="handle"></param>
  /// <returns></returns>
  [DllImport("kernel32.dll", CharSet = CharSet.Auto)]
  public static extern bool CloseHandle(IntPtr handle);

  private WindowsImpersonationContext _windowsImpersonationContext;
  private IntPtr _tokenHandle;
  private string _userName;
  private string _domain;
  private string _passWord;

  const int LOGON32_PROVIDER_DEFAULT = 0;
  const int LOGON32_LOGON_INTERACTIVE = 2;

  /// <summary>
  /// Initialize a UserImpersonation
  /// </summary>
  /// <param name="userName"></param>
  /// <param name="domain"></param>
  /// <param name="passWord"></param>
  public UserImpersonation(string userName, string domain, string passWord)
  {
    _userName = userName;
    _domain = domain;
    _passWord = passWord;
  }

  /// <summary>
  /// Valiate the user inforamtion
  /// </summary>
  /// <returns></returns>
  public bool ImpersonateValidUser()
  {
    bool returnValue = LogonUser(_userName, _domain, _passWord,
        LOGON32_LOGON_INTERACTIVE, LOGON32_PROVIDER_DEFAULT,
        ref _tokenHandle);

    if (false == returnValue)
    {
      return false;
    }

    WindowsIdentity newId = new WindowsIdentity(_tokenHandle);
    _windowsImpersonationContext = newId.Impersonate();
    return true;
  }

  #region IDisposable Members

  /// <summary>
  /// Dispose the UserImpersonation connection
  /// </summary>
  public void Dispose()
  {
    if (_windowsImpersonationContext != null)
      _windowsImpersonationContext.Undo();
    if (_tokenHandle != IntPtr.Zero)
      CloseHandle(_tokenHandle);
  }

  #endregion
}
```



Interview de Cyrille Martraire, fondateur de la communauté Paris Software Craftsmanship

On entend de plus en plus souvent parler du courant Software Craftsmanship, que l'on pourrait traduire « Artisanat Logiciel ». La communauté Paris Software Craftsmanship existe depuis quatre années et compte à ce jour plus de 1000 membres. Rencontre avec son fondateur, Cyrille Martraire, développeur passionné depuis quinze ans et directeur technique de la société Arolla.

Cyrille, pouvez-vous nous expliquer d'où vient le mouvement du Software Craftsmanship et en quoi il consiste ?

A l'origine, « Software Craftsmanship » est le titre d'un livre écrit il y a fort longtemps, mais le courant a véritablement émergé à partir de 2009 quand Robert C. Martin, dit Uncle Bob, a décidé d'en faire son cheval de bataille. En tant que créateur des principes S.O.L.ID., UncleBob est une personnalité incontournable du monde logiciel, une des voix les plus écoutées.

Il a donc popularisé le Software Craftsmanship en réaction à une dérive dans les communautés liées à l'agilité. On est en 2008, tout le monde parle de Scrum et de méthodes agiles du point de vue de la gestion de projets : les Post-it, les rituels tels que le stand up meeting, les itérations, le tracking de vélocité... Mais tout cela sans jamais accorder beaucoup d'importance à la qualité du code produit. Uncle Bob s'insurge et propose alors de remettre l'accent sur les pratiques d'ingénierie logicielle et d'excellence technique : l'agilité est essentielle pour développer efficacement, mais si on néglige les pratiques de développement adéquates, l'agilité permet surtout de développer efficacement de la dette technique. Un de ses slogans est "Build the right thing, and build it right" (Il ne suffit pas de développer le bon outil, il faut aussi bien le développer).

Les quatre principes duSoftware Craftsmanship ont été rassemblés dans le manifeste du Software Craftsmanship, qui reprend et étend le manifeste agile : http://manifesto.softwarecraftsmanship.org/

Ce qui est nouveau dans la vision d'Uncle Bob, c'est la dimension de professionnalisme. Il amène l'idée que le Software Craftsmanship consiste à mettre en œuvre toutes les pratiques connues pour être les meilleures, chacune dans leur contexte, pour écrire et entretenir du code qui soit non seulement économique à produire mais aussi économique à faire vivre demain, après-demain, le mois prochain et même dans cinq ans. Il faut garder à l'esprit que le code que nous produisons aujourd'hui devra être relu, maintenu, corrigé, modifié pour évoluer.

Dans la rubrique professionnalisme du Software Craftsmanship, on trouve aussi l'attitude des développeurs face à leur management. Il est du devoir des développeurs de parfois dire non à certaines décisions lorsqu'elles s'opposent au professionnalisme. Il faut néanmoins préciser que le Software Craftsmanship a pour objectif premier de livrer de la valeur pour les commanditaires. Il ne s'agit en aucun cas d'une rébellion contre le management. Il s'agit au contraire de défendre l'intérêt de l'entreprise et du management dans la durée, même si cela peut parfois paraître en contradiction avec certains objectifs à court terme.

Enfin, la question de la transmission est très importante dans le Software Craftsmanship : il s'agit de développer sa compétence et son professionnalisme, mais aussi de les transmettre autour de nous.

Vous parlez beaucoup de pratiques. Quelles sont celles qu'on peut rattacher au Software Craftsmanship ?

Les pratiques ne sont pas fixes puisque l'idée est de sélectionner les pratiques les plus à jour et les mieux adaptées au contexte. On évite de parler de « meilleures » pratiques, mais plutôt des pratiques les meilleures dans un contexte donné à un instant donné. Aujourd'hui par exemple, Test Driven Development (TDD, le développement dirigé par les tests) est une pratique qui est la plupart du temps celle qui va amener le plus de qualité quand on écrit du logiciel, mais ce n'est pas la seule. Le style de programmation fonctionnelle (FP), les systèmes de types de plus en plus sophistiqués, apportent aussi des avantages. Il faut utiliser les meilleurs outils possibles dans notre contexte.

Lorsque la difficulté n'est pas tant de construire le logiciel que de savoir ce qu'on veut, BDD (Behaviour Driven Development) est une pratique qui professionnalise les recueils de besoins et les rend très efficaces en détectant très tôt d'éventuels problèmes en utilisant au mieux les conversations au travers d'exemples concrets.

D'autres pratiques documentées dans la littérature et qu'on rassemble souvent sous le nom de pratiques de testing et de refactoring legacy (code hérité en français) sont à rapprocher du Software Craftsmanship. L'univers des pratiques est très vaste, on y inclut aussi souvent DDD (Domain Driven Design), les patterns d'architecture, le design émergent, l'architecture flexible, ainsi que les hashtags Twitter comme #NoEstimates ou #NoProject. Ces mots-clés polémiques invitent à réfléchir et à reconsidérer certaines pratiques établies qui ne sont plus forcément pertinentes aujourd'hui dans un univers où nos langages sont plus productifs, et où on doit produire de plus en plus vite des logiciels toujours plus ambitieux.

Pourquoi fonder une communauté Software Craftsmanship ? Quels sont les apports pour les développeurs qui en sont membres ? Que vont-ils y chercher ?

Depuis la création du Paris Jug (Java User Group) en 2008 sous l'impulsion d'Antonio Goncalves, il y eu beaucoup de User Groups qui étaient pour la plupart centrés sur les technologies, Java, .Net, Ruby, JavaScript, ou des frameworks tels que MongoDB ou Neo4J. Il existait bien aussi quelques petits groupes de codage depuis très longtemps, mais ils restaient plutôt confidentiels. Il n'y avait pas de communauté agnostique aux technologies pour permettre aux professionnels de se rencontrer pour discuter des pratiques, apprendre à s'améliorer dans sa façon de coder, de réfléchir aux questions de design, de pratiquer le TDD, ou de prendre du recul sur la façon d'utiliser au mieux les technologies, nouvelles ou non.

Je connaissais par Twitter la communauté Software Craftsmanship de Londres, créée par Sandro Mancuso. Puis, toujours par la magie de Twitter, à l'occasion de la première non-conférence SoCraTes (Software CRAftsmanship and TESting) en Allemagne, j'ai pu entrer en relation avec Sandro qui m'a donné de nombreux conseils pour créer une communauté à Paris. Il est même venu en personne pour faire la keynote de lancement de la communauté parisienne, avec Arolla comme premier sponsor. C'était une prise de risque, mais dès la première soirée on a compris que cette communauté répondait à des attentes fortes. Au fil des rencontres, mois après mois et années après années, un grand nombre de développeurs parisiens ont fait connaissance et ont pu échanger sur des sujets tels que comment tester du code legacy, comment convaincre ses collègues d'essayer TDD ou de

se mettre au pair programming, comment expliquer à mon manager, etc.

Que fait-on pendant les rencontres de la communauté ?

Le format de base est la table ronde. En début de soirée, les participants qui le souhaitent proposent des sujets, puis on se sépare en un, deux ou trois groupes pour en discuter. En général, c'est une simple discussion, parfois on met du code à l'écran. Chacun repart en ayant appris, compris quelque chose, ou partagé un retour d'expérience utile à d'autres. C'est l'opportunité de faire le tour de l'état de l'art sur des sujets variés, de façon rapide, efficace, et surtout sympathique. Et comme, à chaque fois il y en a plus à boire et à manger, on ne perd pas son temps !

La communauté a désormais quatre ans, est-ce qu'on peut dire que c'est un succès ?

C'est un franc succès ! La communauté parisienne est forte de plus de 1000 membres, c'est une preuve d'intérêt. Le format initial accueillait 25 personnes, aujourd'hui c'est en général plus du double. J'ai été rejoint par Stéphane Basnier qui organise la plupart des rencontres cette année. Les entreprises qui désirent devenir sponsors sont aussi plus nombreuses, avec parfois des sociétés assez éloignées du Software Craftsmanship, ce qui montre que le mouvement devient populaire. L'initiative a fait des petits à Lyon et à Marseille par exemple, mais également au sein d'une grande banque française, une communauté Software Craftsmanship interne s'est créée, qui se rassemble tous les mois. Les sujets Software Craftsmanship sont présents aujourd'hui dans la plupart des conférences, la notion d'excellence technique revient sur le devant de la scène et c'est très bien. Beaucoup d'entreprises se mettent au Software Craftsmanship et font appel à des développeurs actifs, en particulier de notre communauté, pour relayer en interne leur envie et leur volonté de re-dynamiser la compétence technique. Scrum ne suffit pas, en revanche avec Scrum et le Software Craftsmanship, les deux bien compris, vous avez la capacité à livrer de la valeur rapidement, loin du bricolage et de l'amateurisme qu'on rencontre encore trop souvent.

Est-ce que la communauté a de l'avenir ?

Oui et non. Elle a de l'avenir en popularité bien sûr car c'est un thème porteur. Le nombre de membres va toujours augmenter, les entreprises vont s'y mettre les unes après les autres. Et comme toute initiative victime de son succès, on va assister à de la récupération, avec des acteurs moins sincères qui vont s'approprier les termes pour des raisons commerciales, jusqu'à ce que le terme perde de son sens. Sur le fond, ces acteurs vont continuer à populariser les idées, mais ce sera regrettable pour les entreprises qui seront parfois déçues, un peu comme dans l'agilité où les coachs agiles ne se valent pas tous. Il faudra de plus en plus faire preuve de discernement dans le casting des intervenants qui se réclament du Software Craftsmanship. ⊞

Et pour aller plus loin...

A lire sur le sujet, le livre « Apprenticeship Patterns » de Dave Hoover, à la genèse du mouvement. Le livre de base sur le Software Craftsmanship est bien sûr le livre d'Uncle Bob, « The Clean Coder ». « Clean Code », un autre livre d'Uncle Bob, est un classique du Software Craftsmanship, mais « The Clean Coder » est un livre plus court, plus léger et d'une lecture fort agréable. Et le dernier en date, un très bon livre, très intéressant, et très personnel, de Sandro Mancuso, sur sa démarche personnelle de Software Craftsman, avec plein de bons conseils, notamment comment recruter un Software Craftsman, comment faire des code reviews, etc. : « The Software Craftsman ».

Développeur/Ingénieur dans la Silicon Valley ?



Olivier Pavie
consultant, formateur, auteur, journaliste dans les nouvelles technologies et la communication
www.opavie.com

Opportunité de carrière ou désir de vouloir trouver un job aux USA ? La Californie, son climat, sa Silicon Valley attirent les profils technologiques du monde entier. Selon le consulat Français de San Francisco, 60000 français seraient installés entre San Francisco et Palo Alto située 60km au sud. Programmez ! est allé enquêter sur les conditions de vie au sens large des ingénieurs développeurs dans cette région où la densité des entreprises technologiques est la plus forte au monde.

Qu'est-ce qui fait de la Silicon Valley qu'elle est la Silicon Valley ?

La Silicon Valley est à la fois un lieu, un état d'esprit, une logique technologique et un environnement propice au développement de projets et d'entreprises. Il s'agit d'un véritable écosystème lié à la technologie qui s'étend de plus en plus vers San Francisco avec plus de 6000 entreprises High Tech. Le nom de Silicon Valley existe depuis 1971 en référence aux sociétés spécialisées dans les processeurs, dont Intel, qui sont venues s'installer et se développer dans la région. Située autour de la ville de Palo Alto à 60 km environ au sud de San Francisco en incluant la rive de la baie, la Silicon Valley est une région de Californie qui occupe 400 km^2. Le cœur de la Silicon Valley est San José, la ville « fondatrice ». Autour d'elle, on retrouve les noms aussi connus et prestigieux que Palo Alto, là où se trouve Facebook, Cupertino, là où est Apple depuis ses débuts, Mountain View, siège notamment de Google, Los Gatos, Menlo Park, etc. Une ligne de train relie l'ensemble des villes de San José jusqu'à San Francisco. Et comme cela est habituel aux USA, 2 autoroutes de 5 voies de chaque côté suivent le même trajet que le train, la 101 et la 280.

Un état d'esprit omniprésent

L'état d'esprit se remarque dès que l'on arrive. C'est simple, c'est marqué dessus ! Les publicités sur les routes regorgent de thématiques liées au business technologique, que ce soit pour des sites d'information, des services Cloud ou de la recherche de talents de développeurs comme en témoigne l'excellente affiche sur les « Hotest Talents » que vous voyez dans ce dossier. Dans la Silicon Valley, tout est technologie. A tel point que dans beaucoup de quartiers, y compris à San Francisco, que ce soit dans la rue ou à côté de vous à table au restaurant, on entend parler de Google, d'Apple, etc. Les Geeks qui se sentent des Aliens en France doivent au moins aller jeter un œil dans cette région. Toutefois, pour ce qui est du travail, c'est en revanche une autre paire de manches ! Il ne suffit pas forcément d'être dans l'informatique. Seuls les profils d'un certain niveau, tireront leur épingle du jeu par une pirouette ou une autre comme le montrent les différents ressentis des ingénieurs et autres interlocuteurs rencontrés lors de notre enquête.

Travailler à tout prix là-bas ?

En plus de l'ambiance, les salaires peuvent attirer puisqu'ils sont d'environ 30% supérieurs à ceux des autres états : de 80 000 à 120 000 $ annuels pour un débutant et des salaires "normaux" autour de 200 000$ pour un ingénieur expérimenté, ces salaires pouvant flirter avec les 250 000 $ à 300 000 $ pour les ingénieurs ou développeurs de génie. Avec le ratio de change Dollars vs Euros, on est sur des salaires du simple au double ou au triple par rapport à la France. Mais ne pas oublier qu'il n'y a pas de charges sociales et qu'il faut payer soi-même sa couverture de sécurité sociale/mutuelle. Pas non plus de retraite "sérieuse". Par ailleurs, comme le dit si bien Alain Raynaud installé depuis 16 ans dans la Silicon Valley, il faut penser à mettre de l'argent de côté... Si vous voulez fonder une famille et offrir des études supérieures à vos enfants, il vous en coûtera 400 000 $ par enfant pour 4 ans d'études supérieures... Très peu de familles dépassent les deux enfants.



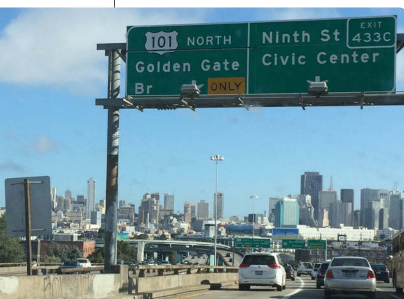
Alain Raynaud,

ingénieur, 16 ans d'expérience

Alain a passé une grande partie de sa vie professionnelle dans la Silicon Valley. Ingénieur Supelec, il est spécialisé dans le silicium, la création de processeurs. C'est d'ailleurs avec la société EVE qu'il s'est installé en 1999. Cette société dont il est cofondateur de la filiale américaine est spécialisée dans le logiciel d'émulation permettant d'architecturer la création de nouveaux processeurs, notamment sur base d'ARM en mode multiprocesseurs. Il s'y est installé avec sa femme et ses enfants. Pour lui, la vie est celle qu'il attendait. C'est lui qui a donné les détails sur les coûts d'une vie de famille repris dans le corps de l'article. "Pour ce qui est de sortir ici, il y a plein de choses à faire mais pour rencontrer d'autres gens, il y a surtout des ingénieurs, alors que sur San Francisco, c'est plus varié." Alain est le fondateur de la Startup Conférence de la SV créée il y a 6 ans.

LE JEU DES VISAS AMÉRICAINS

Pour travailler aux Etats-Unis, il faut un visa ou une green card. De nombreux arrivants Français dans la Silicon Valley arrivent en tant qu'étudiants dans le cadre d'échanges entre universités et grandes écoles. La plupart bénéficient d'un visa J1 voire J2. Ces visas permettent de travailler notamment pour assurer ses frais pendant la période des études; le J1 est également utilisé dans le cadre "Au pair" pour des jeunes de 18 à 26 ans. Le J1 n'est quasiment jamais suffisant pour être embauché par une société High Tech qui va avoir peur qu'un visa résolument adapté ne soit pas obtenu. Or, quand il s'agit d'étudiants, dès que les études sont terminées, il faut avoir été embauché par une compagnie pour demander éventuellement un visa de type H1B, visa qui correspond à un emploi spécialisé exigeant un diplôme de l'enseignement supérieur. Bref, pas toujours facile de jongler... Pour certaines personnes exceptionnelles pour leurs qualités reconnues au niveau national ou international, un visa O peut être obtenu.



En outre, une villa dans la Silicon Valley de 120 m du côté de Mountain View pas loin de chez Google coûte aujourd'hui aux alentours de 1,5 M$! Pour bien comprendre l'attirance du lieu, les maisons se vendent en 8 jours et souvent à un prix supérieur à ce qui était demandé au départ car plusieurs acheteurs se battent et font monter les enchères ! Un bel appartement dans San Francisco capable d'accueillir en colocation trois ou quatre développeurs nouvellement arrivés coûte 6500 à 7000 $ par mois...

Attention, également, si la Silicon Valley attend beaucoup de nouveaux talents, ce n'est malheureusement pas si facile que cela non plus. Il y a des quotas d'immigration, des quotas de visas de travail, et des exigences des employeurs qui augmentent avec le temps pour se garantir que les futurs employés pourront rester. Cela dans deux cas : lorsque le futur employé est en attente de visa et lorsque l'employeur veut lui-même s'offrir des garanties sur la pérennité de l'engagement de l'employé. Dans ce dernier cas, il faut bien comprendre qu'être employé dans la Silicon Valley c'est être employé aux USA, ce qui signifie que vous pouvez être licencié le matin et partir deux heures après, comme vous pouvez vous-même avoir envie de partir ailleurs à peu près dans les mêmes conditions...

Être licencié le matin pour partir l'après-midi ne se passe pas tous les jours non plus... Il y a des raisons à ce genre de situation : soit vous ne faites vraiment pas l'affaire sur un plan personnel et/ou professionnel, soit la société a des difficultés passagères ou définitives qui lui font prendre des décisions très rapides. Là aussi, mieux vaut être bon dans son domaine pour retrouver rapidement un emploi. Vous pouvez aussi anticiper et sauter sur l'occasion qui se présente : ça



Anne-Charlotte Chauvet, *ingénieur, arrivée depuis 2 ans*

Jeune ingénieur Centralienne Anne–Charlotte a été embauchée chez Cumulus Networks, société spécialisée dans l'Open Compute Project qui s'insère dans le mouvement Open Networking : le but étant de créer des switchs à architecture virtuelle; les clients sont principalement les datacenters. Elle a 26 ans, elle est manager Business Operations et a quitté Centrale il y a 3 ans. Elle est arrivée dans la Silicon Valley avec un visa diplomatique dans le cadre d'une action avec Business France (UBIFRANCE). Elle a obtenu 2 fois le visa H1B puis le J1 et a désormais obtenu la fameuse Green Card au mois d'Août 2015. Pour elle, la vie dans la vallée est "plutôt sympa et tranquille, il y a toujours plein de choses à faire avec les concerts, les évènements, etc. L'enfer ce sont les loyers, San Francisco revient cher pour ce que c'est et il faut faire de la colocation, de préférence avec des Français car entre Français il y a une culture de la communauté. Sinon, dans la Silicon Valley, difficile de rencontrer des gens qui soient dans d'autres domaines que les technologies, ce qui n'est pas le cas à San Francisco, où les gens sont plus variés." Anne Charlotte se voit rester là dans les prochaines années, avec "l'ambiance Startup; il y a des nouveaux projets tout le temps, l'environnement est stimulant et les développeurs sont les plus courtisés."

marche dans les deux sens. Vous démissionnez le matin et embauchez ailleurs l'après-midi. Comme anecdote et pour mieux situer le contexte de la guerre technico-économique dans la Silicon Valley, un développeur de génie s'est vu offrir un contrat d'1 M$ sur 3 ou 4 ans s'il rejoignait un nouvel employeur dans la journée ou le lendemain. Ledit développeur ne voulant pas "planter" son employeur actuel a réussi l'exploit de négocier de ne rejoindre le nouvel employeur que 15 jours après : il a pris le risque de ne pas avoir ce contrat ! La morale de l'histoire est quand même que si un employeur a vraiment besoin de vous, il fera aussi quelques "sacrifices".



Sylvain Kalache, *ingénieur, arrivé depuis 7 ans*

Sylvain a obtenu un Master Supinfo. Il est arrivé dans la Silicon Valley il y a 6 ou 7 ans, il a été ingénieur système chez Slideshare pendant 3 ans et est passé chez Linkedin lorsque Slideshare a été absorbé. Pendant ses débuts chez Slideshare et parce qu'il trouvait qu'il manquait un moyen de se réunir entre ingénieurs français dans la Silicon Valley, il a monté l'association While 42 dans le but de réseauter à la fois sur le plan professionnel et sur le plan personnel pour essayer de s'entre-aider dans la vie d'expatrié. Contre toute attente, ça a fait du bruit et bien au-delà de la Silicon Valley. A tel point qu'il y a un mouvement While 42 aujourd'hui dans 40 villes dans le monde. Fort de ce constat et de ses velléités de monter sa propre société, il a lancé sa société TechMeAbroad, un portail de services à destination de tous ceux qui cherchent un emploi à l'étranger et veulent disposer du maximum d'informations et soutien sur le lieu où ils s'installent.

Startups vs grands éditeurs/constructeurs

Quand on est chez Apple, chez Google, Adobe ou la plupart des grands éditeurs, les situations sont généralement stables pour les employés et ce sont surtout eux qui seront appelés par le chant des sirènes de la Startup plutôt que licenciés. Car l'esprit de la Silicon Valley c'est quand même encore et toujours celui du "garage de Steve Jobs et de Steve Wozniak", une émulation qui donne envie de se lancer dans un projet de création de société. Dans ce cadre-là, comme le disent la plupart des développeurs et ingénieurs interrogés, les développeurs sont extrêmement courtisés. Et comment peuvent-ils être si courtisés ? Tout simplement parce que l'argent est omniprésent dans la Silicon Valley. Même s'il n'est pas forcément facile de "décrocher le cocotier", il existe de nombreux moyens de se lancer grâce aux incubateurs et accélérateurs qui ont pignon sur rue et organisent des sessions permettant à celui qui a une idée d'aller jusqu'au bout, ou du moins jusqu'à la présentation de leur projet à des financiers. Les financiers ont des profils variés. Il faut savoir que les incubateurs et accélérateurs ont eux-mêmes de l'argent à investir s'ils trouvent le projet intéressant. Il y a régulièrement des soirées pour faire se rencontrer des investisseurs et des porteurs de projets. Et les investisseurs ne sont pas forcément non plus ceux qu'on croit : il y a les institutionnels du Venture Capitalism (VC), et il y a les compagnies technologiques qui cherchent à investir dans des projets qui leur permettront d'affirmer leur écosystème. C'est ainsi que des sociétés comme Intel, Salesforce, Logitech, HP, SanDisk et bien d'autres ont monté leur département d'investissement que l'on appelle le Corporate Venture, le département chargé de flairer les bonnes opportunités avec des tickets d'entrée de plusieurs millions de dollars.

Toutes ces sources d'argent frais et relativement faciles à obtenir sur des projets sérieux vont pouvoir accroître les perspectives d'emploi avec des embauches à hauts niveaux de salaires pour les meilleurs. Autre tendance de la Silicon Valley, le fait de donner quelques millions de dollars à des fondateurs de startups au bout de deux à trois ans, même sans qu'il y ait eu quoi que ce soit de vendu, pour s'assurer que ces personnes de haute qualité puissent vivre normalement et avoir déjà acquis quelque chose d'ostentatoire dans leur vie professionnelle... Le genre de chose qu'il semble peu probable de voir arriver en France dans les années qui viennent. ⊞



Diego Legrand, *ingénieur, arrivé depuis 2 ans*

Diego est développeur chez Sentient Technologies, société dont le PDG est Français, implantée au cœur de San Francisco. A 26 ans, il est Senior Software Engineer, un des plus jeunes à avoir ce poste dans les anciens de Centrale Paris. Il avait commencé par être développeur sur iOS dans le cadre d'une startup qu'il avait commencé à monter en France alors qu'il travaillait sur du traitement d'image et des algorithmes de machine learning. Lorsque la société Sentient Technologies a été intéressée par son profil, il n'a pas hésité un seul instant. Aujourd'hui son profil est plutôt celui d'un ingénieur que d'un développeur. En termes de vie dans la Silicon Valley, il est ravi d'être à San Francisco plutôt qu'au cœur de la SV. Il est en colocation et va en vélo au travail, quand il a pris sa colocation à 4 dans l'appartement il y a deux ans, c'était pour un tarif de 1300$ par mois par colocataire, maintenant, pour le même appartement, dans les mêmes conditions, il faudrait dépenser 1600 à 1700$! Pour lui, se mettre à la programmation lui a ouvert toutes les portes. Quant à rester sur San Francisco ou rentrer en France, il est très bien là, mais rien ne dit que s'il veut fonder une famille il ne rentrera pas.



Paul Duan, atypique, *arrivé depuis 6 ans*

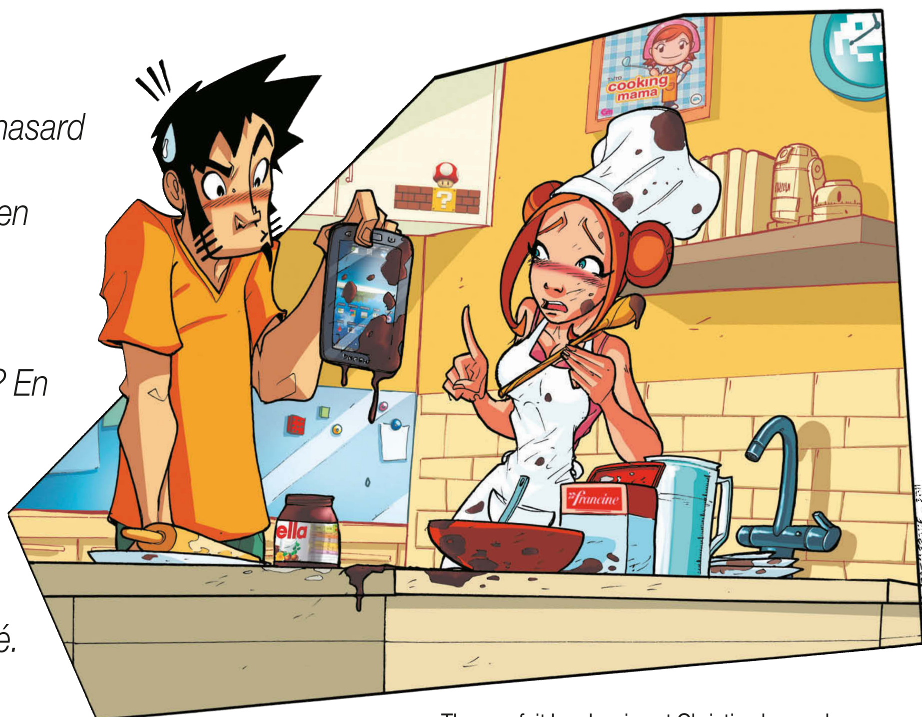
Paul Duan, Français d'origine chinoise élevé dans la banlieue parisienne fait partie de ces gens exceptionnels qui ont obtenu un visa O. La raison ? Il a réussi à entrer à Sciences Po à 16 ans, s'est fait payer son échange avec Berkeley par Sciences Po à 19 ans pour faire des mathématiques (ce qui n'était pas prévu par Sciences Po). Paul est âgé de 24 ans et vient de créer l'ONG "Bayes Impact" (NDLR, Bayes en raison du Théorème de Bayes) pour sauver des "millions de personnes" après s'être spécialisé dans les algorithmes de traitement des Big Data et avoir mis en pratique ses talents dans le Machine Learning chez Eventbrite pendant presque deux ans. L'objectif de Bayes Impact est de disrupter le fonctionnement des institutions. Pour en arriver à créer sa propre ONG dans la Silicon Valley, il a participé à un des multiples accélerateurs de Startup, l'un des plus prestigieux, le Y Combinator et a pu mettre en place un financement de départ. Il travaille actuellement sur un projet pour le Ministère Americain de la Santé visant à réduire le nombre de réadmissions à l'hôpital liées à de mauvais diagnostics avant la sortie du patient.

Les Geeks : le monde geek à découvrir en BD...



La rédaction

Nous l'avons découvert un peu par hasard (merci à César pour nous l'avoir fait découvrir...). Comment le geek vit-il en société, en entreprise, à la maison ? Comment se passe une relation amoureuse et sa passion pour la technologie ? Est-il un peu sectaire ? En réalité, il y a le geek et le nerd (même si les deux ne sont pas incompatibles). Régulièrement, il dépanne les copains et quand des geeks se rencontrent, la conversion peut être surréaliste pour le non-initié.



Tout ce cocktail donne une infinie de situations drôles, cocasses et que l'on pourrait croire irréelles. Mais elles sont souvent bien réelles et nous en avons vécu de très nombreuses.
A découvrir avec les auteurs. Le prochain album sortira courant novembre (en principe).

Comment est née la BD "Les Geeks" ?

C'est Delphine et Melanyn, les 2 filles de GANG qui en ont eu l'idée. Elles travaillaient dans la BD et vivaient avec des geeks à la maison, et Soleil (notre éditeur) recherchait justement des albums "thématiques". Elles ont donc enrôlé les geeks qui les entouraient pour former l'équipe de GANG (3 hommes et elles deux) au scénario, puis ont cherché un dessinateur et un coloriste adaptés. Thomas Labourot avait un style qui nous plaisait bien et qui collait à l'univers mais quand on l'a recruté, on ne savait pas encore qu'il passait la moitié de sont temps à jouer aux jeux vidéos, construire des scènes complètes de star-wars en lego ou collectionner des figurines de mangas !

Chaque personnage à un caractère geek bien défini, est-ce du vécu ou ce que vous aviez observé ?

C'est surtout une façon pour nous d'aborder toutes les facettes de la "geekitude" car il y en a beaucoup. On a posé les 5 personnages principaux (Julie, Fred, Charline, Vince et Hubert) au début de la série avec chacun leurs caractéristiques geeks : Hubert était le nerd de la bande, Vince plus branché ciné/comics, Fred plutôt JV et informatique, Charline une mac addict et Julie...survit au milieu de la bande. Arnold et CB se sont rajoutés plus tard pour couvrir des profils plus variés.

De quelle manière travaillez-vous les personnages et les scénarios des albums ? On sent une profonde culture geek avec de nombreuses références.

On regarde plein de trucs au cinéma et sur Internet, on lit des livres et des BDs, on joue à des jeux, on bidouille du code et des machines et puis...pouf pouf, des idées surgissent. A un moment on fait des enfants aussi, et ça donne de nouvelles idées de gags. Le fait d'être nombreux et d'avoir chacun ses domaines de prédilection est un atout pour se renouveler. On essaye aussi de rester dans l'actu, de toujours parler des nouveaux univers ou des nouvelles technologies qui peuvent intéresser les geeks. Tous les albums commencent par une réunion de brainstorm. On boit des coups en racontant des bêtises, on a plein d'idées très drôles. Certains prennent des notes. Au bout d'un moment on est fatigués et on va se coucher. Le lendemain on essaye de déchiffrer les notes et on fait le tri. On réalise que certaines idées ne sont plus drôles à jeun et on les laisse tomber. On garde celles qui restent et on les approfondit pour en faire des gags.

Finalement qui fait quoi dans l'équipe ?

Les 5 membres de gangs contribuent au scénar en fonction de leur disponibilité et inspiration. Thomas fait les dessins et Christian les couleurs. Les idées pour la couverture et la quatrième de couverture sont souvent issues de looooooongues discussions entre tout ce petit monde.

Comment définissez-vous le geek (le vrai geek) ? N'est-il pas un peu nerd ?

Le geek, pour nous, est quelqu'un de passionné. Que se soit pour des univers fantastiques, du code, du retrogaming ou des jeux de plateau...c'est quelqu'un qui cherche à découvrir, approfondir, partager et contribuer.

Le tome 11 sortira dans quelques semaines. Quelques spoilers à nous dévoiler ?

Du cul. On a découvert que ça faisait vendre, alors on va en mettre dedans. Achetez le! Plus sérieusement, on a voulu faire évoluer un peu les rapports entre les personnages, parce qu'on sait qu'au bout d'un moment ce sont aussi eux qui doivent porter la série et que nos lecteurs aiment suivre.

Est-ce difficile de se trouver de nouvelles histoires et situations après 7 ans de BD ?

Oui. C'est pour ça qu'il nous faut 1 an pour trouver 36 idées pour faire un album complet. En plus on vieillit et on commence à devenir hermétiques à certains sujets. Par exemple, aucun gag sur Minecraft dans les geeks car aucun des scénaristes n'y joue (je suis pas sûr qu'on ait même compris a quoi ca servait ce truc). Et puis comme la majorité des scénaristes, on a un « vrai » boulot à côté, on prend notre temps...



0234.012345789 Recursivité





0234.012345789 Geeks Commandements





0234.012345789 Ted et Bill ou quoi ?



Vis ma Vie by Xebia



Au sein d'une entreprise, quel que soit son secteur d'activité, se côtoient des employés aux responsabilités très diverses. Ce constat est particulièrement prégnant dans les cabinets de conseil où les métiers des fonctions transverses sont souvent méconnus des consultants. Il est pourtant essentiel, pour construire une société pérenne de mettre en place une organisation participative et de s'appuyer sur l'intelligence collective ; en un mot, d'échanger.



C'est dans cet esprit que Xebia s'est livrée à l'exercice du "Vis Ma Vie". Pendant une journée, tous les postes de management ont été occupés par une ou un consultant(e) : Président, COO, CTO, Directrice Administrative et Financière, Directrice Recrutement, Directrice Marketing, Directrice Commerciale, Directeur Xebia Studio et Consultant Manageur. Puis ce sera au tour de chacun des Directeurs de prendre la place des consultants en allant vivre leur vie durant une journée.

Travailler ensemble

Une entreprise est constituée de salariés aux prérogatives qui peuvent, de prime abord, paraître contradictoires voire antagonistes. De l'administratif au commerce en passant par le marketing, la finance, le management et les opérations, chaque manageur se voit confier des responsabilités qui sont souvent ignorées ou incomprises par les autres.

Néanmoins, tout le monde partage un objectif commun : faire grandir l'entreprise.

Chacun contribue à sa manière et en fonction de son rôle à cette oeuvre collaborative, en faisant vivre le "Why" (la raison d'être) de l'entreprise.

Pour fluidifier cette mécanique, il est important que tout le monde entrevoie et comprenne les contraintes et impératifs de chacun.

Ce projet "Vis Ma Vie" s'inscrit naturellement dans les valeurs et la culture de la société, cabinet de conseil et réalisation IT basé à Paris. Le partage de connaissance, une des valeurs fondatrices de la société, est inscrite dans son ADN. La communication se fait en toute transparence dans une perspective d'écoute active et d'amélioration continue.

Les "Vis ma Vie"

Cette expérience ludique a donc été proposée aux consultants de Xebia pour leur permettre d'expérimenter d'autres métiers au sein de l'entreprise. Cette expérience leur a permis d'appréhender les responsabilités, contraintes et tâches de chaque fonction, et de partager largement, sans censure, les leçons apprises pour mieux vivre ensemble.

C'est au cours de son XKE (Xebia Knowledge Exchange, journée mensuelle dédiée au partage de la connaissance) du mois de mai, que l'ensemble des Xebian(e)s ont été invités à postuler pour l'un de ces 9 postes : Président, COO, CTO, Directrice Administrative et Financière, Directrice Recrutement, Directrice Marketing, Directrice Commerciale, Directeur Xebia Studio et Consultant Manageur. Ce Vis ma Vie a permis aux Xebian(e)s sélectionné(e)s de découvrir un autre pan de l'entreprise et de participer à toutes les décisions. Pas de tabous, pas de secrets, toutes les informations leur ont été accessibles !

Puis, ce sera au tour des 9 directeurs de se confronter au quotidien des consultants en prenant leur place durant une journée.

Ceux qui ont vécu ces Vis ma Vie proposeront leur retour d'expérience à l'ensemble des Xebian(e)s durant notre séminaire annuel en Octobre pour que tout le monde découvre les faces cachées de chaque métier !

Xebia a également souhaité partager cette expérience plus largement au travers d'une Web série à découvrir sur vismavie.xebia.fr.



Xebia est un cabinet de conseil Parisien spécialisé dans les technologies Big Data, Cloud, Web, les architectures Java et mobilité dans des environnements agiles. Les Xebians partagent leurs connaissances au quotidien via notre blog (blog.xebia.fr), twitter (@XebiaFr) et participent/animent régulièrement des conférences et des Techevents.



Niels, CTO Digint Solutions (Abu Dhabi) : « *Enchanté, cher lecteur !* »

Aujourd'hui on m'a demandé de faire un petit retour d'expérience sur ma vie de geek, alors voilà. Je m'appelle Niels Freier, j'ai 28 ans et je développe depuis l'âge de 13 ou 14 ans. De manière plus générale, je suis surtout passionné par l'outil informatique depuis mon tout premier ordinateur, reçu à l'âge de 8 ans.

Après mes études, j'ai été consultant en développement puis architecture sur les technologies Microsoft. Suite à ces quelques années riches en expérience, je me suis orienté vers les technologies Cloud mais surtout le *Big Data* au sein d'un cabinet de conseil qui m'a permis de beaucoup voyager entre Paris et les USA. Une bonne année, certes. Mais cela ne m'a pas suffit. J'avais pour projet de créer ma propre entreprise sur les Emirats Arabes. C'est chose faite, maintenant que je suis installé dans mon rôle de CTO à Digint Solutions. Nous développons des solutions pour les entreprises et gouvernements qui permettent de contrer les personnes visant à attenter à la sécurité nationale.
Je suis maintenant résident de l'Emirat d'Abu Dhabi depuis bientôt 1 an et très content de mon choix malgré les difficultés liées à l'expatriation dans un pays si différent, et le fait de monter une entreprise à partir de rien.

Comment es-tu tombé dans l'informatique et plus spécialement dans le développement ?

Comme une bonne partie des *geeks*, je suis tombé dans l'informatique quand nous avons eu notre premier ordinateur à la maison. C'était un cadeau de mon père pour moi et mon grand frère. Déjà il avait compris depuis longtemps que l'informatique allait être une part importante de notre vie donc quand les finances l'ont permis, il nous a équipé. Et comme à peu près tous les enfants, la première et probablement seule chose que nous avons faite dessus pendant longtemps fut de jouer. J'ai encore le souvenir du plaisir d'insérer une disquette et après les quelques secondes de chargement, de voir les pixels s'animer sous nos yeux. À chaque fois un moment magique ! Après quelques années – oui je dois le reconnaître : je n'ai pas attaqué le développement dès le tout début, je ne savais tout simplement pas que cela existait – j'ai tout de même compris qu'il était possible et surtout pas vraiment difficile de donner des ordres à l'ordinateur par le biais de commandes somme toute assez simples : *if*, *else*, *then*... Rien de sorcier, mais quels pouvoirs ! Je me suis donc plongé, cette fois-ci pour de bon, de longues soirées dans le code. Déjà les plus petits programmes enfantins me faisaient passer pour un roi. Vous savez bien de quoi je parle; par exemple le jeu où il faut deviner un chiffre où le rôle du programme est de répondre si c'est plus ou moins grand que la valeur définie par l'utilisateur... Eh oui nous avons tous eu nos débuts. Les années passant, j'ai continué de progresser tout en suivant les nouveautés du monde informatique. Vous comprendrez tout aussi bien que moi qu'une update de mon Pentium I en Pentium III était nécessaire, même indispensable pour faire tourner Age of Empire 2. Oui, j'admet avoir continué de jouer tout de même. Ce qui m'a valu de longues discussions avec mon père avant de l'obtenir ce Pentium III. J'ai profité de l'arrivée du nouveau Pentium et de l'Internet pour faire mes premières armes sur le Web. C'est un outil merveilleux qui m'a permis, comme tellement d'autres personnes, de pouvoir échanger et apprendre d'une façon si différente de celle d'un élève écoutant un professeur tout en essayant de ne pas s'endormir en classe. Attention anecdote : j'ai même réussi à prendre un zéro pour triche en 3ème après avoir distribué sur le réseau de l'école un CMS en php permettant de générer un site configurable ayant permis a toute la classe de produire un petit exercice de construction d'une première page Internet. Ce jour là, malgré le zéro, j'ai réellement compris que l'informatique et plus généralement le développement pouvaient me (nous !) permettre d'aller plus loin, de casser des barrières depuis longtemps établies. Le développement, c'est virtuel. À l'inverse d'une chaîne de fabrication dans une usine de voiture par exemple. Oui le développement ne réclame rien, si ce n'est un ordinateur. Maintenant que c'est un outil disponible à peu près partout, il est clair que le mouvement de création de programmes ne va pas s'arrêter et va continuer d'ouvrir de belles portes.

Pour toi, qu'est ce qui fait que l'on aime toujours et encore le développement, la technique ?

Pour la même raison qu'expliquée plus haut. Le fait que cela bouge, évolue en permanence rend



impossible de rester passif ou fainéant dans ce domaine sans être dépassé en quelques mois. Mais surtout pour le côté créatif. Que faisions nous quand nous étions enfant ? Des châteaux de sables, des forts pour contrer la monté de la mer et j'en passe. Et bien le code pour moi c'est un peu pareil. Il a ce côté créatif qui force à trouver la meilleure solution pour obtenir quelque chose de performant et robuste.

Tu as gardé un regard très geek : gadget, veille techno, c'est important pour ton job et ta passion ?

Mon job a consisté pendant plusieurs années à être consultant en développement, puis consultant *Big Data*. Alors oui, la veille est une part indispensable de mon travail. Maintenant que je suis CTO, cela reste nécessaire d'être toujours au courant des nouveautés du milieu. Non pas qu'il faille se jeter sur toutes les nouveautés qui semblent intéressante. Vraiment pas. Mais il faut savoir les suivre, les évaluer et pourquoi pas les adopter si le besoin s'en fait sentir. Après tout, je suis et resté un *geek* alors oui je craque souvent pour des Kickstarters qu'au final je n'utiliserais qu'une fois le jour de la réception du colis. Oui j'aime mélanger différents appareils et les faire communiquer par le biais d'une API standard ou en forger une toute nouvelle pour l'occasion. Mais cela reste du loisir, de petits *side-projets* et nous savons tous qu'ils ne mènent que bien rarement à quelque chose d'autre qu'une bonne rigolade entre potes ou simplement une tranquille soirée de code avec une bonne musique.

Tu es parti à Abu Dhabi pour monter ta start-up. Comment as-tu mûri l'idée et ton envie de partir ? Quelles qualités, penses-tu, faut-il avoir pour cette nouvelle vie ?

Mon projet de monter une entreprise sur Abu Dhabi, et plus généralement aux Emirats

Arabes et dans le Golf date d'il y a 3 ou 4 ans. Au cours d'une mission chez un de mes clients de l'époque, j'ai rencontré celui qui est mon *business partner* sur ce projet.
Nous avons muri ensemble ce projet pendant plusieurs mois, avons fait une première tentative non concluante et puis mis en sommeil le projet pour quelques temps. Mais la graine était plantée et cela n'a jamais vraiment quitté nos esprits. Enfin quand un soir j'ai reçu un coup de fil de Dubaï me demandant si j'étais prêt a lâcher ma vie sur Paris pour venir m'installer sur Abu Dhabi et enfin monter notre projet, j'ai réfléchi quelques minutes, surtout pour savoir comment j'allais pouvoir conclure les affaires courantes avant de donner ma réponse : ok, j'arrive ! Les Emirats ne sont pas une destination facile, ni classique dans notre milieu des startups et autres compagnies technologiques.

Comment se prépare une telle aventure : famille, amis, budget, installation sur place ?

Cela dépend vraiment de la situation de chacun avant le départ. Dans mon cas ce fût plutôt simple. Quand j'ai reçu le coup de fil de mon ami sur les Emirats me proposant de le rejoindre pour enfin monter notre projet, j'ai fait tourner l'idée dans ma tête pendant quelques minutes puis j'ai annoncé que je poserais ma démission le lendemain. La partie compliquée a été prise rapidement. L'idée, c'est de ne pas trop réfléchir je pense. Si l'envie est là alors il faut le faire ! Ensuite vient la partie plus pratique de l'expatriation : s'occuper des papiers, donner son préavis pour son logement (dans le cas d'un départ a l'étranger un mois suffit !) et bien sûr annoncer la nouvelle à tout le monde, amis comme famille. Dans mon cas, la destination quelque peu exotique a fait son effet. Dans notre milieu, il est commun d'avoir des amis aux US, Londres, Berlin, etc. Mais bien peu sur les Emirats. Donc oui l'effet de surprise était là, sauf bien sûr pour ceux qui connaissaient déjà mon projet de longue date mais ils n'étaient pas nombreux. Une fois le mois de préparation terminé, j'ai fini mon dernier jour dans mon rôle de consultant *Big Data* sur Paris un vendredi soir et j'ai embarqué le dimanche matin pour ma nouvelle vie, sans avoir aucune idée de où j'allai dormir le soir. Mais sincèrement, arrivé à ce point là, c'était un détail. Une fois sur place, il a fallu s'occuper de l'ensemble des démarches administratives pour obtenir un visa de résidence, un visa de travail, un permis de conduire, le transfert des parts de la société à mon nom et j'en passe. L'ensemble des démarches a du me prendre environ un mois. En général à courir partout le matin pour les papiers et l'après-midi occupé à construire le projet en lui-même. En termes de Budget. Encore une fois cela va dépendre de chacun, mais l'ensemble de mes frais de départ a été pris sur le solde tout compte de mon poste parisien. N'ayant pas pris de vacances durant une année complète, cela m'a permis de gérer le départ sans trop de soucis. Pour la suite de l'aventure et le temps d'encaisser nos premiers contrats, il a fallu vider petit a petit l'épargne. C'est le prix à payer pour tenter sa chance et monter une entreprise, une sorte de pari !

Quel bilan tu peux tirer de ton expérience depuis ton arrivée à Abu Dhabi, il y a 11 mois ?

Je pense que le syndrome des montagnes russes est connu de l'ensemble des entrepreneurs. Ce sentiment que de jour en jour, d'heure en heure la situation change du tout au tout. Un matin on se lève en se disant que c'est bon, aujourd'hui est LA journée qui va lancer le business pour au final se coucher déprimé comme jamais en se disant qu'il ne reste plus qu'a refaire sa valise et rentrer. Mais on résiste et le lendemain on recommence jusqu'au jour où cela se débloque enfin réellement, dans un sens ou dans l'autre. Sur le plan de la vie personnelle, je dirais que l'expérience est fantastique. Que de nouvelles choses et relations. Pourtant comme beaucoup d'entrepreneur ma vie sociale se résume à assez peu de temps libre. Cependant, le simple fait de découvrir un pays si différent, une autre culture, tellement de cultures différentes, que même ce peu de temps est déjà un grand bond en avant sur le plan personnel.
Professionnellement justement c'est quelque chose de nouveaux. Du très bon et du très mauvais à la fois. On fait face à de nouveaux challenges, on se retrouve dans le rôle du chef qui ne sait pas s'il aura de quoi finir le mois. Cela permet de relativiser pas mal de chose et il faut avoir les nerfs solides pour tenir le coup. Mais en même temps, être CTO d'une entreprise, et qui plus est dans un domaine tel que celui de Digint Solutions, est quelque chose de vraiment fantastique, que ce soit sur le plan technique, métier mais également humain.

Te vois-tu revenir en France ou t'installer définitivement dans un autre pays ?

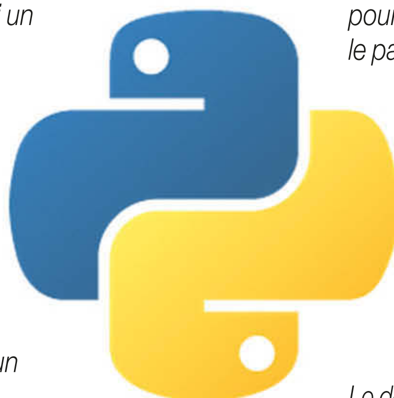
Revenir en France j'y pense souvent. Au moins une fois par semaine. À chaque fois que les choses tournent mal ici. Mais c'est vraiment les seuls moments où j'y pense vraiment, car autrement je ne regrette en rien ma décision de départ. Oui bien sûr, comme je suppose presque tous les français expatriés, j'aime la France. Mais est-ce une raison pour ne pas partir quelques années, voir un peu plus ? Non, je pense que le monde actuel nous permet bien des choses. Avant, disons il y a une centaine d'années, une expatriation était le projet d'une vie et sans doute sans possibilité de retour. Maintenant tout est possible et c'est devenu quelque chose de commun et de facile.
Quant au futur, je n'ai encore rien fixé. Encore bien trop tôt pour cela. Mais après les Emirats, que ce soit d'ici quelques mois ou quelques années, je pense tenter ma chance dans un autre pays. Dans le meilleur des cas, j'espère avec une filiale de mon entreprise actuelle par exemple. Mais nous verrons bien à ce moment là. Un pays plus classique ou encore une destination excentrique ? Hmmm, difficile à dire, je n'aime pas faire comme tout le monde ;).

Quels conseils pourraient donner à celles et ceux qui te lisent ?

Prenez votre temps ! Mais une fois que la chance se présente, ne vous posez pas trop de question et foncez. En début de carrière cela peut être un bon tremplin, et quitte à aller apprendre un nouveau langage de développement, autant en profiter pour booster votre Anglais, Croate, Chinois, Arabe, Espagnol, Tagalog, etc. Un peu plus tard, une fois votre profil technique renforcé, vous aurez le choix de continuer purement dans cette voie, celle permettant de prendre son pied sur du code toute la journée ou bien avoir LA bonne idée qui pourrait vous permettre de prendre des parts de marché sur un créneau porteur. Dans ce cas là, même si le choix le plus simple reste de le faire là ou vous êtes, surtout si votre marché est global, par exemple dans le cas d'une application Web/mobile, songez à vous rapprocher de votre marché en optant pour l'expatriation. Plus personne ne pourra vous dire que vous n'avez pas osé sortir de votre zone de confort ! Apres mes études, je suis resté une journée par semaine professeur dans mon ancienne école, Epitech. Une des choses que j'ai toujours transmise à mes étudiants (et beaucoup me détestaient, si si je le sais), c'est que quoi que vous fassiez, il faudra toujours faire face a des gens qui seront contre vous. Que ce soit le prof qui cherche à démontrer que votre projet n'est pas stable, vos binômes qui vous utiliseront et j'en passe. Si j'ai bien un conseil à donner, c'est de faire face, de vous mettre en position de force et d'avancer jusqu'a votre objectif. Autant que possible, faites tout cela de manière propre. Personne ne doit jamais pouvoir rien vous reprocher dans le futur. Ainsi votre succès sera pleinement mérité. ⊞

"I may not have gone where I intended to go, but I think I have ended up where I needed to be."
Douglas Adams,
The Long Dark Tea-Time of the Soul

Python, au-delà du serpent



Python n'est pas qu'un serpent. C'est aussi un langage de programmation. Il était jusqu'à présent relativement discret en France, mais depuis 3-4 ans, il sort de sa pénombre. Et c'est une bonne nouvelle.

Python est un langage apparu en 1990. Il fut créé par Guido van Rossum comme langage de script de haut niveau pour le système Amoeba. C'est un langage interprété, multiplateforme. Il s'agit aussi d'un langage objet pour des développements critiques et exigeants. Sa syntaxe est une de ses forces. Il est aussi open source et sa licence est compatible GPL. Le succès du langage s'est lentement mis en place. Et la France accueille aussi son événement Python : PyData, orienté données.

En 2001, la fondation Python a été créée (Python Software Foundation) pour gérer l'écosystème et le langage en lui-même. Mais cela n'a pas empêché une coupure dans le monde python en 2008 avec la sortie de Python 3, incompatible avec Python 2.x même si le langage reste en grande partie identique. Python 2 est vu comme un langage pour les applications existantes et pour maintenir le patrimoine applicatif, mais cette branche ne connaîtra plus d'évolutions majeures.

Une partie du succès de Python est son apprentissage aussi bien par les développeurs, que les étudiants et les enfants. Les nombreuses bibliothèques et extensions permettent d'étendre le langage.

Le développeur Python n'est pas un profil si répandu en France et les salaires ne sont pas forcément plus élevés que la moyenne. En Amérique du Nord, le profil Python est recherché et en automne 2014, il affichait le 3e salaire moyen, pour le développeur, le plus élevé avec 100 717 $ bruts (source : Quartz)...

Dans ce numéro, nous vous proposons ce dossier Python pour revenir sur les bases du langage, Python et les données, quelques fonctions avancées et des retours de développeurs Python.
Que du bonheur avec Python

La rédaction

Python : les secrets d'un succès

Le langage de programmation Python a été créé au début des années 1990 par Guido Van Rossum (GvR) afin de faciliter le développement de logiciels par les chercheurs du CWI aux Pays-Bas qui utilisaient des bibliothèques de calcul écrites en C. Le succès a été au rendez-vous, puisque 25 ans plus tard Python est dans le top 5 des langages de programmation, que l'on prenne comme référence le TIOBE Index (http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html) ou le classement IEEE Spectrum (http://spectrum.ieee.org/static/interactive-the-top-programming-languages-2015).



Laurent Pierron
(laurent.pierron@inria.fr)

Le langage est resté plutôt stable, la version 2 est sortie en 2000 et la version 3 en 2008, cette dernière rompt la compatibilité ascendante avec la version 2, mais des modifications mineures sur les programmes Python 2 permettent de les faire fonctionner en version 3.
Python est un langage de programmation de type impératif, dans lequel un programme est une suite d'instructions qui modifient la mémoire de l'ordinateur sur lequel il est exécuté. Mais un programme Python peut également utiliser les paradigmes de programmation fonctionnelle et de programmation objets. Les entités manipulées en Python sont des objets sans aucune exception; le typage de ces objets est fort et dynamique (défini à l'exécution). Python est un langage interprété avec une machine virtuelle. Python dispose d'une interface de programmation (API) en C, qui lui permet d'être enrichie avec des bibliothèques écrites en C.
Depuis son lancement, Python a été enrichi de nombreuses bibliothèques, qui couvrent pratiquement tous les domaines de la création de logiciel : vous trouverez sûrement le module qui répond à vos besoins. Cependant pour exploiter au mieux ces bibliothèques, vous devez acquérir une bonne compréhension du fonctionnement de Python. Heureusement celui-ci est plutôt simple, car la syntaxe est proche des langages algorithmiques. Les éléments de structuration du code sont en petit nombre.
Le modèle objet est général, du type simple comme un entier au module, qui définit un ensemble de classes et de fonctions; tout est objet. Dans cet article, nous allons essayer à travers des exemples, de couvrir de manière non exhaustive, les principaux aspects du langage qui font sa particularité, sa puissance et sa notoriété.

SYNTAXE DU LANGAGE

Plongeons dans Python

Pour vous permettre de comprendre à quoi ressemble un programme Python 3, nous allons développer un logiciel qui liste le vocabulaire utilisé dans un livre et les occurrences d'apparition de chacun des mots de ce vocabulaire. Nous allons donc créer un programme ***mots.py***; il est de coutume de suffixer les noms de programmes Python par '***.py***'.
Un programme Python commence par un texte de documentation facultatif, qui s'étend sur plusieurs lignes le voici :

```
"""Affiche la liste des mots d'un texte.

Le nom du document à analyser est demandé à l'utilisateur.
La liste des mots est triée par nombre croissant d'apparitions.
Le nombre de mots et la taille du vocabulaire sont aussi affichés.
"""
```

Afin de mettre au point notre logiciel au fur et à mesure de son écriture nous allons utiliser l'interpréteur interactif fourni avec le langage, que nous pouvons évoquer comme tel depuis la ligne de commande :

```
$ python
Python 3.4.3 |Continuum Analytics, Inc.| (default, Mar  6 2015, 12:07:41)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Dans l'interpréteur, nous allons pouvoir charger (*importer*) le programme que nous avons écrit, et tester quelques unes de ses propriétés, notamment son *type* (c'est un objet de type *module*) et visualiser sa *documentation* :

```
>>> import vocabulaire
>>> type(vocabulaire
<class 'module'>
>>> help(vocabulaire)
Help on module vocabulaire:

NAME
    vocabulaire - Affiche la liste des mots d'un texte.

DESCRIPTION
    Le nom du document à analyser est demandé à l'utilisateur.
    La liste des mots est triée par nombre croissant d'apparitions.
    Le nombre de mots et la taille du vocabulaire sont aussi affichés.
```

Maintenant nous allons écrire un peu de code, en commençant par une fonction qui demande à l'utilisateur un fichier à lire :

```
def demande_fichier(prompt=""):
  """Demande un nom de fichier et retourne son descripteur."""
  nom = input(prompt).strip()
  try:
    fichier = open(nom)
  except IOError:
    print("Veuillez taper le nom d'un fichier lisible")
    fichier = demande_fichier(prompt)
  return fichier
```

Ce code nécessite quelques explications. Une fonction en Python se déclare par le mot-clé ***def*** suivi d'un espacement, puis du nom de la fonction et de ses paramètres formels entre parenthèses et en fin du caractère **' :'**, qui désigne un début de bloc de code.

Ici le nom de la fonction est ***demande_fichier,*** et elle prend un paramètre formel dont le nom est ***prompt*** qui a une valeur par défaut "" (chaîne de caractère de longueur 0). Si la fonction doit retourner une valeur, elle le fait par l'appel à l'instruction ***return***; il peut y avoir plusieurs ***return*** dans le corps d'une fonction, ou aucun si celle-ci effectue uniquement des effets de bord comme afficher une valeur à l'écran.
Une fonction qui tout simplement double un nombre peut s'écrire en une ligne :

```
def double(x): return 2*x
```

Dans notre fonction, on trouve une seule instruction ***return*** à la fin de la fonction, cette instruction renvoie le descripteur du fichier ouvert en lecture, ce qui assure son existence.
Je vous ai dit que le bloc de description de la fonction commençait après le caractère **' :'**, la première ligne de code doit être décalée vers la droite par rapport à la ligne de déclaration de la fonction, toutes les autres lignes doivent être alignées à cette première ligne. Une ligne dont le premier caractère est aligné avec le mot-clé ***def*** indique à Python que le bloc de code est terminé. Cette syntaxe force une bonne lisibilité du code Python, car celui-ci ne peut fonctionner correctement que s'il est bien aligné. Ne vous inquiétez pas, la majorité des éditeurs de code savent vous aider à aligner du code Python.
Maintenant, venons-en au code de la fonction. Comme vous l'avez deviné, la première ligne est la documentation de la fonction, c'est la même syntaxe que la documentation du module.
La seconde ligne, qui commence par ***'nom ='*** ressemble à une affectation dans les langages de programmation classiques, mais c'est différent, et c'est une particularité de Python. Lorsque l'interpréteur Python rencontre une instruction avec le symbole **'='**, il évalue l'expression dans la partie droite et la range en mémoire dans un objet dont il associe l'adresse à la partie gauche de l'équation, c'est-à-dire à la *variable* ***'nom'*** dans notre code source. Chaque nom (ou *variable*) en Python, contient donc une référence vers un objet rangé dans la mémoire. Vous pouvez tester facilement cela en utilisant l'interpréteur :

```
>>> x = 6    # On affecte la valeur 6 à la variable x
>>> id(x)    # Demande l'adresse mémoire associée à x
4297327392
>>> id(6)    # Demande l'adresse mémoire associée à 6
4297327392   # '6' est aussi un objet, même adresse que x
>>> id(double) # Demande l'adresse mémoire associée au type double
4322057480    # un type est aussi un objet, eh oui !!!
```

Et voilà, vous connaissez maintenant toute la magie de Python !
Ah si! Il reste une chose que l'on verra en détails plus loin, Python a deux types d'objets : les objets non modifiables (nombres, chaînes, tuples) et les objets modifiables (pratiquement tout le reste).
Maintenant nous allons évaluer l'expression à droite de l'équation (`nom = input(prompt).strip()`) de la manière suivante :

- D'abord appel de la fonction `input` avec le paramètre `prompt`, qui affiche le texte passé en paramètre, attend une ligne de texte puis retourne un objet de type texte ;
- Sur l'objet de type texte est appelé la méthode `strip()`, qui enlève les espaces en début et fin de texte. Le point avant `strip()` indique à Python d'appeler une méthode ;
- `nom` finalement contient l'adresse du nom de fichier saisi par l'utilisateur sans espaces, en début et en fin du texte.

Continuons l'analyse de notre code à partir de la ligne **try:**, qui est l'instruction Python pour encapsuler un bloc de code qui pourrait contenir une erreur à l'exécution, ce code est contenu dans la ligne

suivante, en l'occurrence c'est la tentative d'ouvrir (open()) un fichier (nom) en lecture (par défaut), le descripteur du fichier sera associé à la *variable* fichier. S'il est impossible d'ouvrir le fichier le bloc de code après l'instruction **except IOError**: est exécuté.
Classiquement, le type de l'erreur attrapée est indiqué, il peut y avoir plusieurs erreurs à rattraper sur la même ligne, mais également plusieurs instructions except. Le code de gestion de l'erreur consiste simplement à afficher un message (**print()**) et à demander à nouveau un nom de fichier par appel récursif de la fonction demande_fichier.
La dernière ligne de la fonction en dehors du bloc **try**: est une instruction (**return** fichier), qui renvoie le descripteur de fichier vers l'expression appelante.
On peut tester la fonction programme sous l'interpréteur :

```
>>> import vocabulaire
>>> f = vocabulaire.demande_fichier("=> ")
=> bbb
Veuillez taper le nom d'un fichier lisible
=> bovary.txt
>>> f
<_io.TextIOWrapper name='bovary.txt' mode='r' encoding='UTF-8'>
>>> f.read(40)
'Gustave Flaubert\nMADAME BOVARY\n\n\n(1857)\n'
```

On charge le fichier *vocabulaire.py*, on appelle la fonction **demande_fichier** dont on gardera le résultat dans la *variable f*. Ensuite, on affiche la représentation du fichier et on lit les 40 premiers caractères du document.

Lecture du document

Ajoutons les quatre lignes suivantes à notre programme pour lire le contenu de notre livre :

```
# Lecture du texte contenu dans le document demandé
# et transformation en minuscules
with demande_fichier('votre livre -> ') as ftext:
    texte = ftext.read().lower()
```

Les lignes commençant par le symbole # sont des commentaires. Un symbole # peut-être placé à tout endroit sur une ligne, et tout ce qui suit jusqu'à la fin de la ligne est un commentaire.
La troisième ligne est une instruction **with** … **as**, qui permet, entre autres, de s'assurer que le fichier (ftext) ouvert sur la ligne du **with** sera fermé après son utilisation. Une seule expression permet de lire le contenu complet du fichier et de le convertir en lettres minuscules : ftext.read().lower(). Comme vous vous en doutez texte pointe sur l'objet texte créé par l'expression. Maintenant on arrive à la partie traitement du document, d'abord on va découper le texte en mots après avoir fait un nettoyage pour enlever les caractères qui ne sont pas des lettres :

```
# Remplacement des caractères non alphabétiques par des espaces
texte = ''.join(x if x.islower() else ' ' for x in texte)
# Découpage du texte en mots
mots = texte.split()
```

La première expression après 'texte =', un peu obscure pour un néophyte, est typiquement idiomatique de Python. Cette expression signifie de coller (''.join) tous les x si x est une lettre minuscule (**if** x.islower()) sinon mettre un espace (**else** ' ') pour tous les caractères x dans texte(**for** x **in** texte).
En Python, il est possible dans une expression de parcourir une liste ou tout autre objet se comportant comme une liste. La deuxième instruction découpe (split()) le texte en utilisant les espaces. La variable mots pointe vers un objet qui est une liste de mots. Un petit exemple sous l'interpréteur :

```
>>> txt = "plus de spam désolé"
>>> mts = txt.split()
>>> mts
['plus', 'de', 'spam', 'désolé']
```

Une liste est un objet, qui contient d'autres objets placés séquentiellement. La création et la visualisation d'une liste consiste en des objets séparés par le symbole virgule (**,**) entourée des crochets ouvrants et fermants **[]**. Les objets de la liste n'ont pas à être tous du même type, ces objets peuvent aussi être des listes. Une liste est en fait un tableau à une dimension de taille variable. Une liste, contrairement aux chaînes de caractères, peut être modifiée, en changeant un élément, en ajoutant ou supprimant des éléments.

Calcul des statistiques

La liste est un des deux types de conteneurs standards de Python les plus utilisés, il sert à représenter des collections ordonnées, des tableaux, des listes, mais aussi des arbres. Nous allons voir maintenant le second type de conteneur très utilisé, le dictionnaire :

```
# Création d'un dictionnaire pour enregistrer
# les occurrences de chaque mot
dico = dict.fromkeys(mots, 0)
for mot in mots:
    dico[mot] += 1
```

Nous souhaitons associer à chaque mot du texte le nombre d'apparitions de ce mot. Il faut donc associer à chaque mot un nombre. Pour associer un objet à une chaîne de caractères, il existe un type de conteneur Python bien pratique : le **dictionnaire** (dict). Un dictionnaire permet d'associer à un objet non mutable un autre objet. Dans la littérature de la programmation, le dictionnaire se nomme aussi tableau associatif ou table de hachage. Voici un exemple de dictionnaire dont les clés sont des chaînes de caractères et les objets associés de différents types (caractères, nombre, liste) : {'nom': 'Zorg', 'objets': ['masse', 'sabre'], 'vie': 35}. Un dictionnaire est une liste encadrée par des accolades ({}) de paire ***clé : objet*** séparées par des virgules. Dans notre programme nous créons d'abord le dictionnaire à partir de la liste des mots (mots) grâce à une méthode (fromkeys()) sur le type dictionnaire (dict). Dans un second temps nous parcourons (**for** mot **in** mots:) la liste des mots et incrémentons de 1 (dico[mot] += **1**) le nombre associé au mot dans le dictionnaire.
L'itération la plus originale en Python est le parcours de liste avec l'instruction : **for** x **in** liste:. Cette formulation très naturelle permet de parcourir une liste d'objets, elle est même utilisée pour des itérations numériques en utilisant la liste des nombres à parcourir, par exemple : **for** i **in** range(0,10):. permet de faire une boucle avec un indice i allant de 0 à 9 par pas de 1. L'autre instruction d'itération que nous ne verrons pas est : while condition :. Qui permet de répéter un bloc d'instructions tant que la condition est vraie.

Affichage du résultat

On a fini le calcul du nombre d'occurrences, il nous reste à effectuer un affichage lisible du résultat. On va un peu compliquer l'affichage, car on souhaite la liste des mots d'au moins trois caractères dans l'ordre inverse du nombre d'occurrences. Nous voulons que les mots soient alignés à droite et les nombres aussi, nous aurons besoin de la largeur du mot le plus long.

```
# Affichage de la liste des mots dans l'ordre croissant d'occurrences
largeur = max(map(len, mots))  # Largeur du mot le plus long
```

```
patron = "{{0:>{largeur}}} : {{1:5d}}".format(**locals())
for (mot, occ) in sorted(dico.items(),
                  key=lambda item: (item[1], item[0])):
  # Utilisation de largeur pour présenter le résultat
  if len(mot) > 3:  # Ignore les mots courts
    print(patron.format(mot, occ))
```

La première équation nous permet de calculer la largeur du mot le plus grand, en calculant la longueur (len) de chaque mot et en prenant le maximum (max). La fonction map permet d'appliquer une fonction à chaque élément d'une liste et retourne la liste des résultats. Cette fonction est classique des langages fonctionnels, auxquels Python emprunte des éléments permettant de rendre le code plus concis et expressif. Vous aurez remarqué ici que Python autorise des fonctions en passage de paramètre à une autre fonction, c'est normal une fonction est un objet comme un autre. Je vous l'ai déjà dit en Python pratiquement **tout est objet**, y compris les types et les fonctions.
La ligne suivante (patron =) permet de construire une chaîne de formatage qui sera utilisée pour l'affichage du résultat en utilisant la plus grande largeur. Je n'expliquerai pas ici la syntaxe de ce format, mais je me permets de vous faire remarquer cette syntaxe étrange : **locals() qui offre la possibilité de passer en paramètre d'une fonction un dictionnaire (**), en l'occurrence les associations variable et valeur définies en local (locals()) dont la variable largeur calculée précédemment. Vous pouvez alors voir que cette *variable* **largeur** est utilisée dans la chaîne de formatage. Pour s'en convaincre, jouons un peu avec l'interpréteur :

```
>>> largeur = 7
>>> patron = "{{0:>{largeur}}} : {{1:5d}}".format(**locals())
>>> patron
'{0:>7} : {1:5d}'
```

Vous remarquerez que largeur a bien été remplacé par 7.
On arrive presque à la fin. Il ne nous reste plus qu'à parcourir la liste des mots dans l'ordre croissant des occurrences. Pour cela on trie la liste des paires (mot, occurrences) du dictionnaire dico par ordre croissant en utilisant la fonction sorted de Python, qui peut prendre une clé (key=) de tri en paramètre. La clé de tri est une fonction qui, à un objet à trier, associe une clé; ici ce sera une fonction anonyme (**lambda**), qui permutela paire (mot, occurrence). A noter que dico.items() permet de retourner la liste sous forme de paire (clé, objet) des éléments d'un dictionnaire. Vous noterez dans l'itération **for** l'utilisation d'un appariement ((mot, occ)) pour extraire de la paire le mot et son nombre d'occurrences. Ce type de paire est appelé un ***tuple*** en Python, le ***tuple*** se généralise à un nombre quelconque d'objets entre parenthèses, s'utilise comme des listes, mais est immutable. Le ***tuple*** est fréquemment utilisé pour renvoyer plusieurs valeurs en retour d'une fonction. Enfin la dernière instruction est une instruction conditionnelle (**if**), qui affiche par la fonction **print** les mots formatés (.format(mot, occ)) avec le formatage patron.
Voici le programme complet :

```

"""Affiche la liste des mots d'un texte.

Le nom du document à analyser est demandé à l'utilisateur.
La liste des mots est triée par nombre croissant d'apparition.
Le nombre de mots et la taille du vocabulaire sont aussi affichés.
"""

# Fonction de demande d'un nom de fichier
def demande_fichier(prompt=""):
  """Demande un nom de fichier et retourne son descripteur."""
  nom = input(prompt).strip()
  try:
    fichier = open(nom)
  except IOError:
    print("Veuillez taper le nom d'un fichier lisible")
    fichier = demande_fichier(prompt)
  return fichier

# Lecture du texte contenu dans le document demandé
# et transformation en minuscule
with demande_fichier('votre livre -> ') as ftext:
  texte = ftext.read().lower()

# Remplacement des caractères non alphabétiques
# par des espaces
texte = ''.join(x if x.islower() else ' ' for x in texte)
# Découpage du texte en mots
mots = texte.split()
# Création d'un dictionnaire pour enregistrer
# les occurrences de chaque mot
dico = dict.fromkeys(mots, 0)
for mot in mots:
  dico[mot] += 1

# Affichage de la liste des mots dans l'ordre
# croissant d'occurrences
largeur = max(map(len, mots))  # Largeur du mot le plus long
patron = "{{0:>{largeur}}} : {{1:5d}}".format(**locals())
for (mot, occ) in sorted(dico.items(),
                  key=lambda item: (item[1], item[0])):
  # Utilisation de largeur pour présenter le résultat
  if len(mot) > 3:  # Ignore les mots courts
    print(patron.format(mot, occ))
# Quelques statistiques
print("vocabulaire / mots : {} / {}".format(len(dico), len(mots)))

```

Structuration des programmes

Pour structurer un programme écrit en Python on dispose de deux autres techniques en plus des fonctions : les classes et les modules.

Classes

Les classes permettent de grouper des comportements (méthodes, attributs), qui seront communs à un ensemble d'objets. Les classes permettent d'ajouter des nouveaux types à Python. Une classe est elle-même un objet de la classe 'type' avec ses méthodes et ses attributs. Pour vous montrer les particularités de mise en œuvre des classes en Python, le mieux est de jouer avec un exemple simple de taxinomie animalière (*classes.py*), que voici :

```
class animal:
  def __init__(self):
    self.vivant = True
  def mourir(self):
    self.vivant = False

```

```
7   class mammifere(animal):
8       allaite = True
9       def mettrebas(self):
10          if self.vivant:
11              return self.__class__()
12
13  class oiseau(animal):
14      pondeur = True
15
16  class ornythorinque(mammifere, oiseau):
17      pass
```

De la ligne 1 à 5, nous définissons une classe de nom **animal**, l'instruction **class <nom>**: permet de définir une nouvelle classe. Dans cette classe nous définissons deux méthodes avec la même syntaxe que pour définir une fonction.

La méthode **__init__**(), a une sémantique particulière en Python; elle sera appelée au moment de la création d'une nouvelle instance de la classe. Il n'est pas obligatoire de la définir, mais elle servira principalement à initialiser les attributs de l'objet ainsi créé. Il existe de nombreuses méthodes (*__str__*, *__eq__*, *__sizeof__*, etc.) ayant une sémantique propre à Python, qu'il est important de connaître afin que les programmes que l'on développe se comportent d'une manière attendue. Par rapport à d'autres langages à objets, ces méthodes définissent des *interfaces*, même si il n'y a aucune distinction entre des interfaces et des classes en Python, laissant une grande simplicité au langage.

La méthode **mourir**(), permet de modifier l'attribut vivant de l'objet.

Ces deux méthodes ont un argument de nom self, cet argument est en fait l'objet lui-même qui est passé à la méthode. Ce nom self est une convention. Tout autre nom peut être utilisé, mais le programme est plus compréhensible si self est utilisé pour désigner le nom de l'objet.

Les méthodes d'instance et les méthodes de classe se distinguent par le fait que les méthodes d'instance ont en premier argument l'objet lui-même. Maintenant, jouons un peu dans l'interpréteur avec ce premier exemple :

```
>>> from classes import animal
>>> type(animal)
<class 'type'>
>>> bob = animal()
>>> type(bob)
<class 'classes.animal'>
>>> bob.vivant
True
>>> bob.mourir()
>>> bob.vivant
False
```

Dans ce petit exemple vous pouvez constater que la classe animal est un objet de la classe 'type'. La création d'une instance se fait en utilisant une classe comme une fonction. L'accès à un attribut d'objet se fait en utilisant le point (on l'a déjà vu dans l'exemple précédent) bob.vivant et l'appel d'une méthode toujours en utilisant le point et en ajoutant des parenthèses pour appeler la fonction : bob.mourir().

En Python une classe peut *hériter* d'une autre, c'est-à-dire disposer des attributs et de méthodes de la classe parente et avoir ses propres attributs et méthodes. Dans l'exemple la classe **mammifère** est définie en sous-classe de **animal** en plaçant le nom de la classe parente dans les arguments de la nouvelle classe créée. La classe **mammifère** a un attribut allaite valable pour toutes les instances. Vous remarquerez le bloc de code de la méthode **mettrebas**, qui permet de créer un nouvel objet de la même classe si l'attribut vivant de l'objet appelant est vrai. __class__ est un attribut disponible pour tous les objets, qui donne la classe de l'objet (comme la fonction type()) et comme on veut créer un nouvel objet il suffit d'ajouter les parenthèses () pour faire appel au constructeur d'instance, à noter que l'on aurait pu écrire aussi : type(self)().

Jouons un peu avec cette nouvelle classe, on crée un mammifère, on vérifie qu'il allaite et on lui crée un petit :

```
>>> zoe = mammifere()
>>> zoe.allaite
True
>>> zig = zoe.mettrebas()
>>> type(zig)
<class 'classes.mammifere'>
```

Jusque-là tout va bien, continuons notre taxinomie, nous souhaitons ajouter la famille des ornithorynques, qui est un animal à la fois mammifère et oiseau (pas si vrai que cela, mais c'est pour l'exemple), donc il doit hériter des propriétés des mammifères et des oiseaux.

L'héritage multiple va nous permettre cela, en créant la classe **ornythorinque** qui hérite à la fois de mammifère et oiseau :

ornythorinque(mammifere, oiseau)

Jouons un peu avec cet animal étrange, en créant une ornithorynque Fifi qui va donner naissance à un bébé Riri, qui est aussi un ornythorinque :

```
>>> fifi = ornythorinque()
>>> fifi.vivant, fifi.allaite, fifi.pondeur
(True, True, True)
>>> riri = fifi.mettrebas()
>>> type(riri)
<class 'classes.ornythorinque'>
```

Je vous invite à consulter la documentation Python pour en savoir plus sur l'algorithme de recherche dans la hiérarchie de classes.

Modules

Un autre moyen de structurer les programmes Python est le module.

Un module est un fichier Python, qui contient des définitions de fonctions, de classes, des instructions *d'affectation* et toutes autres instructions. Nous avons donc déjà créé deux modules dans cet article : mots.py et classes.py. Un module est un objet comme un autre, il s'intègre dans un autre module par les instructions import <module>, import <module> as <nom> ou from <module> import <objet>, qui permet de sélectionner les objets à importer.

Un module peut-être divisé en sous-modules en les répartissant dans une arborescence de répertoires et de fichiers. Le nom du répertoire est alors le module et les fichiers dans le répertoire sont des sous-modules.

Eco-système autour de Python

L'interpréteur Python dispose d'un dévermineur interactif (pdb), mais l'élément le plus important pour la mise au point des programmes est le système d'affichage de la pile d'exécution particulièrement parlant, par exemple, on a eu une division par zéro dans la fonction f lors de l'appel à cette fonction à la ligne 2 :

```
$ python bug.py
Traceback (most recent call last):
  File "bug.py", line 2, in <module>
    print(f(0))
  File "bug.py", line 1, in f
```

```
def f(x): return 1/x
ZeroDivisionError: division by zero
```

De nombreux environnements de développement existent pour Python, dont certains sont écrits en Python comme Tkinter (fourni avec la distribution Python). Par ailleurs la syntaxe de Python est connue de la majorité des éditeurs de code, certains comme Emacs ayant des modes intelligents pour aider l'utilisateur à jongler avec la tabulation, qui sert à la structuration du code Python. La distribution standard de Python est fournie avec de nombreuses bibliothèques (batteries incluses), mais il existe d'autres distributions adaptées pour du développement scientifique ou des applications Web. Pour compléter les bibliothèques standards, il existe un dépôt officiel (Pypi : http://pypi.python.org/) accessible par la commande pip. La version principale de Python est écrite en C et en Python, mais il existe une version Java (Jython) et une version .NET (IronPython). Python dispose également d'outils pour créer la documentation (pydoc), pour tester le code (doctest, unittest) et pour le distribuer (distutils). Afin d'améliorer le code et de détecter des éventuels erreurs avant l'exécution, il existe plusieurs analyseurs de code statique de fort bonne facture, qui renforcent également les bonnes pratiques : pylint, pyflakes

Domaines d'applications

Python est utilisé dans de nombreuses universités et également dans les lycées pour initier les élèves à l'algorithmique et la programmation. Le principal avantage est sa syntaxe proche du langage algorithmique, sans être obligé d'ajouter d'informations supplémentaires. On trouvera également des environnements de simulation comme VPython (visualisation 3D) et PyGame (réalisation de jeux vidéos), faciles d'accès. Vous pouvez apprendre par vous-même avec des MOOCs sur Python, dont un en français sur la plateforme FUN (https://www.france-universite-numerique-mooc.fr/courses/inria/41001S02/session02/about).

Python est utilisé efficacement dans de nombreux domaines d'applications, à la fois comme langage complémentaire mais aussi comme langage principal d'une application, grâce aussi à sa bonne intégration avec les principales bases de données (PostgreSQL, MySQL, SqlLite). Le tableau suivant, en partie tiré du site officiel Python, liste quelques exemples d'applications dans les domaines où Python s'illustre particulièrement :

Domaines	Logiciels
Programmation Web	Django, Plone, Zope, Pyramid, Bottle, Tornado, Flask, web2py, Twisted, MoinMoin
ERP	ERP5 basé sur Zope, Odoo (OpenERP + Tiny ERP)
Développement de GUI	wxPython, tkInter, PyGtk, PyGObject, PyQt
Calcul Scientifique et Numérique	SciPy, NumPy, Pandas, IPython, Sage, Matplotlib, Pyx
Génie logiciel	Buildbot, Trac, Roundup, Scons, Mercurial
Administration systèmes	Ansible, Salt, OpenStack

Un domaine spécifique dans lequel Python réussit bien est l'automatisation de tâches dans des logiciels complexes comme *Blender* ou *LibreOffice*. Il existe des bibliothèques de bon niveau et bien maintenues pour traiter à peu près tous les types de données : textes (pygmentize, nltk), images (PIL), son et musique (wave, audioop, pythoneon), molécules (Python Molecular Viewer), etc. Un domaine où Python n'est pas vraiment adapté, c'est celui des systèmes embarqués où le code doit être au plus proche du matériel, mais il est utilisé pour la programmation à un niveau supérieur par exemple dans les Raspberry (Raspberry-pi) et les robots Poppy (pypot) et il avait été choisi par Nokia comme langage de programmation des smartphones de la série 60.
Voir aussi : https://www.python.org/about/success/

Avantages

Python est un langage très facile à maîtriser en raison du petit nombre d'éléments qui le composent : types de base classiques et immuables (entier de longueur quelconque, double, complexe, booléen, chaîne de caractères, tuple), 2 collections à tout faire (liste, dictionnaire), 14 instructions (8 usuelles) pas une de plus, 5 structures de contrôle (if, while, for, try, with) et 3 éléments de structuration (fonction, classe, module) et c'est tout.
Les programmes écrits en Python sont en général concis, mais lisibles.
Le développement est facilité par l'utilisation de l'interpréteur interactif, qui permet de tester du code sans avoir à créer un environnement spécifique.
Il existe en général une ou plusieurs bibliothèques qui répond à vos besoins, il ne vous reste plus qu'à assembler les composants ensemble, des composants qui, s'ils sont bien écrits, se comportent comme les objets de base que vous connaissez déjà. Par exemple en Python ce sont les mêmes méthodes pour lire un fichier sur le disque ou sur Internet, même si l'accès n'est pas le même.
Le typage est fort, ce qui permet d'éviter des erreurs de transtypage à l'exécution, Python vous signalera qu'il ne peut pas faire telle opération sur tel objet.
Python s'intègre très bien avec des bibliothèques écrites en C, il existe même des outils automatiques pour créer les interfaces en Python.
Les codes Python apportent une grande généricité et une forte réutilisation grâce à plusieurs mécanismes :

- Création dynamique d'objet ;
- Recherche de méthodes et d'attributs apparentés à des interfaces ;
- Fonctions, classes et modules sont des objets comme les autres et peuvent être utilisés comme des fonctions, ou passés en paramètres d'autres fonctions ;
- Fermeture transitive : une fonction peut être créée avec un contexte set à sa création dynamique d'objet, autorise une forte généricité et réutilisation des codes développés.

Inconvénients

Le principal inconvénient est celui qui fait sa force ; c'est le typage dynamique, qui implique qu'il n'est pas possible de vérifier avant exécution de manière sûre un programme, ni de connaître la signature des fonctions et des méthodes. Python est très lent, 10 à 100 fois plus lent qu'un programme écrit en C, car interprété et dynamique, mais il est possible dans certains cas d'améliorer les performances avec cython ou pypy. Le passage de paramètres par adresse peut engendrer des effets de bord indésirables, en modifiant des valeurs non souhaitées.
Python détermine automatiquement la portée des variables (local / global) suivant l'utilisation, toutes les variables en partie gauche d'une affectation sont locales, il faut donc utiliser le mot clé **global** ou **nonlocal** pour déclarer la liste des variables globales modifiées dans un sous-bloc, sous peine d'une erreur à l'exécution ou au masquage non souhaité d'une variable globale par une variable locale de même nom.
Python est mal adapté pour effectuer du parallélisme à base de *thread*, car un seul *thread* peut être utilisé à la fois, par contre il est possible de faire du parallélisme local et distant en utilisant plusieurs instances de Python communiquant (multiprocessing, iPython).

Conclusion

J'espère que ce petit tour d'horizon sur les principes de Python et ses domaines d'application vous aura donné envie d'en savoir plus sur ce langage de programmation au succès bien mérité et qu'il fera bientôt partie de votre panoplie de développeur. 

Automatiser les tâches du quotidien avec IPython



Benoit Lacherez,
chargé de mission e-éducation à l'agence départementale Numérique 64. Utilisateur de Python depuis 2000.
http://lacherez.info

Un adage bien connu chez les administrateurs système dit : "Si tu dois faire un travail une fois, fais-le à la main ; si tu dois le faire deux fois, fais un script". Il y a beaucoup d'avantages à cette méthode :
si j'ai à faire une tâche fastidieuse (comme des copier/coller multiples), je préfère passer une heure à écrire un script qu'une demi-heure à faire ces opérations ingrates,
car alors, je garde une trace de ce que j'ai fait, j'apprends, je m'amuse et surtout... je gagne de précieuses minutes pour la prochaine fois où je devrai les faire.

Ecrire des scripts rapidement et de façon itérative avec IPython

Quand on n'est pas engagé dans un « vrai » travail de développement, mais qu'on écrit des petits scripts vite faits pour régler les problèmes du quotidien, il est important de ne pas passer trop de temps à concevoir la logique du programme : il s'agit souvent de manipuler des volumes de données limités, avec des opérations simples. Aucune place pour les optimisations complexes et les algorithmes astucieux.

IPython (pour « interactive Python », http://ipython.org/) est un interpréteur interactif qui offre (parmi beaucoup d'autres) des fonctions indispensables pour un tel usage : complétion, utilisation comme shell et historique.

La **complétion** fonctionne de façon standard en pressant la touche **<Tab>**.

L'**utilisation comme shell système** est elle aussi très simple : un certain nombre de commandes Unix de base (**pwd**, **ls**, **cd**...) sont accessibles directement sous forme de “commandes magiques” (même sous Windows). Les autres sont disponibles en ajoutant "**!**" en début de ligne. Il est donc possible de travailler sans sortir d'IPython.

Mais ce qui fait vraiment la différence, c'est la **gestion de l'historique**. Comme vous pouvez le voir sur la capture d'écran, chaque commande est numérotée, il est donc facile de la retrouver, pour la relancer (**%rerun <numéro(s) de commande(s)>**) ou pour l'enregistrer dans un fichier (**%save <nom de fichier> <numéro(s) de commande(s)>**).

Un exemple simple

Pour illustrer cette démarche, voici un exemple simple : nous voulons envoyer un courrier électronique à quelques dizaines de personnes dont les adresses sont dans un fichier texte (une adresse par ligne). Notre script va écrire toutes ces adresses sur une seule ligne, séparées par des virgules, pour pouvoir copier/coller la ligne dans une messagerie.

Dans IPython, commençons par ouvrir le fichier **adresses.txt** et lire toutes les lignes dans une variable **adresses**, puis définissons le séparateur à utiliser :

```
In [1]: with open("adresses.txt") as f:
   ...: lignes = f.readlines()
In [2]: SEP = ","
```

Créons ensuite une liste **adresses** qui contient toutes les lignes sans le caractère de fin de ligne, puis assemblons ces adresses avec le séparateur dans la chaîne **destinataires** :

```
In [3]: adresses = [l[:-1] for l in lignes]
In [4]: destinataires = SEP.join(adresses)
In [5]: destinataires
Out[5]:
'spam@goo.fr,truc@bidule.ch,,machin@spam.fr,foo@mmail.com'
```

Une des adresses est vide, à cause d'une ligne vide dans le fichier de départ. Corrigeons notre commande de création de la liste **adresses** (en faisant plusieurs fois **<flèche vers le haut>** pour l'afficher) :

```
In [6]: adresses = [l[:-1] for l in lignes if l!="\n"]
```

Essayons à nouveau de créer notre chaîne de destinataires (pour retrouver la ligne correspondante dans IPython, saisissons **dest** puis **<flèche vers le haut>**, jusqu'à la bonne) :

```
In [9]: destinataires = SEP.join(adresses)
In [10]: print(destinataires)
```

Le résultat nous convient, nous pouvons maintenant créer un script en enregistrant dans **ligne_adresses.py** les commandes 1 à 10 de ma session :

```
In [11]: %save ligne_adresses.py 1-10
```

La commande **%save ligne_adresses.py 2 1 5 8-10** aurait permis de ne conserver que les lignes utiles, mais dans un cas comme celui-là, il est plus pratique de tout enregistrer et de supprimer les lignes dans un éditeur de texte.

Désormais, après modification, nous avons un script qui peut être lancé avec la commande **%run** :

```
In [12]: %run ligne_adresses.py
spam@goo.fr,truc@bidule.ch,machin@spam.fr,foo@mmail.com
```

Cet exemple est évidemment très simple (et même simpliste), mais il montre une démarche facile pour construire un script par étapes.

Avec la bibliothèque standard de Python et les nombreuses bibliothèques tierces que l'on peut trouver, beaucoup de tâches peuvent être automatisées : opérations sur les fichiers, récupération d'informations sur le Web, extraction de données avec des expressions régulières... Les seules limites sont celles de votre imagination ! ⊞

```
benoit@benoit-370R4E: ~
benoit@benoit-370R4E:~$ ipython3
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
Type "copyright", "credits" or "license" for more information.

IPython 1.2.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: # Afficher le répertoire courant (commande magique)

In [2]: pwd
Out[2]: '/home/benoit'

In [3]: # Afficher les processus en cours (commande système)

In [4]: !ps
  PID TTY          TIME CMD
 5648 pts/28   00:00:00 bash
24626 pts/28   00:00:00 ipython3
26118 pts/28   00:00:00 sh
26119 pts/28   00:00:00 ps

In [5]:
```

Python Data Tools

Python est un langage de programmation interprété et orienté objet de plus en plus populaire, en particulier dans le monde de la Data Science. Cet article introduit les bases de la programmation en Python pour des applications autour de la donnée ainsi que les principales librairies, NumPy, Pandas et Scikit-learn.



Yoann Benoit,
Data Scientist chez Xebia



Installer Python et ses principales librairies

Pour installer Python efficacement avec toutes les principales librairies nécessaires, le mieux est de s'orienter vers Anaconda, qui est une distribution gratuite de Python incluant près de 200 packages et contenant des installers à la fois pour Python 2.7 et 3.4.
En plus d'IDE comme Spyder ou même IntelliJ Idea, un IPython Notebook est disponible. Il permet, de manière interactive et rapide, l'exploration et la manipulation de données. Vous pouvez grâce à ce notebook combiner l'exécution de code, de textes, de formules mathématiques (LaTeX) et de graphes en un seul document. Pour le lancer, il suffit de taper les commandes suivantes dans votre console dans le bon répertoire suite à l'installation d'Anaconda :

```
>> ipython notebook
```

Le gestionnaire de notebooks devrait alors apparaître dans votre navigateur. Bien que les principales librairies soient déjà installées avec Anaconda, il est possible que vous ayez besoin d'installer un package qui ne soit pas encore présent. Pour ce faire, il faut utiliser la commande *conda* ou *pip* comme suit :

```
>> conda install myPackage
>> pip install myPackage
```

Comme expliqué en introduction, nous allons vous présenter au cours de cet article trois des librairies les plus importantes pour l'exploration, l'analyse et de traitement de données en Python. Les deux premières, que sont NumPy et Pandas, vont être utilisées principalement pour charger, explorer et préparer la donnée en vue d'analyses plus poussées futures. La librairie scikit-learn entre ensuite en jeu afin d'utiliser efficacement des algorithmes de Machine Learning sur les données préparées au préalable.

Le calcul scientifique avec NumPy

NumPy est le package essentiel pour le calcul scientifique (l'autre référence est SciPy) et l'analyse de données en Python. De nombreuses autres librairies de plus haut niveau, notamment pour le Machine Learning, sont basées sur ce package. Ses principales fonctionnalités sont les suivantes :

- Création et manipulation extrêmement performante d'arrays à n dimensions permettant des opérations arithmétiques vectorisées ;
- Opérations mathématiques rapides sur un tableau de données sans nécessité d'écrire des boucles ;
- Algèbre linéaire ;
- Possibilité d'intégration de code écrit en C, C++ et Fortran.

Le package NumPy ne fournit pas de fonctionnalités haut niveau pour l'analyse de données, celles-ci sont plutôt présentes dans les librairies qui sont basées dessus. Il est cependant fortement bénéfique de maîtriser ses concepts pour utiliser au mieux les autres outils.
Il est nécessaire d'importer au préalable la librairie pour avoir accès à ses fonctionnalités, souvent via une appellation donnée.

```
>>> import numpy as np
```

Création de tableaux multidimensionnels

L'objet ndarray (N-dimensional array) de NumPy permet d'avoir de larges tableaux de données, qui doivent être du même type, sur lesquels on peut effectuer de nombreuses opérations mathématiques. Il est très simple des les créer, tout comme d'effectuer des opérations :

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]]) # Create a 2x3 Array
>>> type(a) # ndarray
```

Pour accéder ou modifier un ou des éléments du tableau, il suffit de spécifier leur position entre crochets.

```
>>> print a[0,0] # Print one element
>>> print a[:,1] # Return the second column
>>> a[1,:] = [7,8,9] # Replace second line elements by [7, 8, 9]
```

En créant un tableau de cette manière, si l'on ne spécifie pas de type pour les données, NumPy va tenter d'inférer le meilleur possible.
Il est de plus possible de créer des tableaux spéciaux : remplis de zéros, de uns ou bien de random.

```
zeros = np.zeros(5) # One dimensional array of 0s
ones = np.ones(shape=(3, 4), dtype=np.int32) # 3x4 Array of 1s
random_array = np.random.randint(0, 10, (3, 4)) # 3x4 Array with random integers between 0 (included)
 and 10 (excluded)
```

D'autres fonctions permettent la création de ndarray, comme *arange, empty, eye* et d'autres.

Opérations sur des tableaux

L'avantage indéniable des tableaux NumPy est qu'ils permettent de faire des opérations de manière vectorisée, c'est-à-dire sans avoir à faire de boucle for.
Toutes les opérations arithmétiques sur des ndarray s'appliquent à tous les éléments.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]]) # 2x3 Array
>>> a*2 # Every element of the array is multiplied by 2
>>> a*a # Element-wise multiplication (this is not a matricial operation)
```

De la même manière, de nombreuses fonctions mathématiques usuelles sont disponibles dans NumPy et applicables sur des ndarray.

```
>>> np.sqrt(a) # Element-wise square root
```

```
>>> np.exp(a) # Element-wise exponential value
```

Beaucoup d'autres fonctions sont disponibles, telles que abs, floor, cos, etc. Des fonctions statistiques sur les tableaux sont accessibles en tant que méthodes de ces derniers.

```
>>> a.sum() # Sum of all elements of the array
>>> a.mean() # Mean of all elements of the array
>>> a.sum(axis=0) # Sum by column
>>> a.min(axis=1) # Min by row
```

Si l'on veut faire de l'algèbre linéaire sur les tableaux, quelques fonctions sont disponibles.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]]) # 2x3 Array
>>> b = np.array([[1, 2], [3, 4], [5, 6]]) # 3x2 Array
>>> np.dot(a,b) # Matrix multiplication
```

Comme expliqué précédemment, NumPy est donc une librairie extrêmement puissante pour le calcul scientifique, mais ne fournit pas de fonctionnalités haut niveau. Elle constitue le socle principal pour d'autres librairies avec une syntaxe plus simple pour la manipulation et l'exploitation de données.

La manipulation de données avec Pandas

NumPy est un outil efficace mais peu pratique pour manipuler des données hétérogènes. Dans la pratique, la librairie Pandas, basée comme beaucoup d'autres sur NumPy, est plus adaptée, car elle propose des structures de données haut niveau et de nombreux outils pour la manipulation efficace et rapide de données. Pandas possède les fonctionnalités suivantes :

- Des structures de données avec des axes labélisés ;
- Manipulation de séries temporelles ;
- Gestion efficace des données manquantes ;
- Opérations relationnelles semblables au SQL.

```
>>> import pandas as pd
```

Il existe deux principales structures de données dans Pandas, *Series* et *DataFrame,* permettant de très nombreuses manipulations de données.

Series

Une Series est un tableau à une dimension (≈ un vecteur) de variables labélisées de n'importe quel type (integers, strings, floating point numbers, Python objects, etc.). L'axe des labels d'une Series se nomme l'index. Pandas va créer un index par défaut lors de la création de la Series, bien que les labels puissent aussi être spécifiés.

```
>>> s1 = pd.Series([1,3,5,np.nan,6,8]) # A Series containing integers and one non-specified value (nan)
>>> s2 = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
```

Chose qui n'était pas possible de manière simple avec NumPy, les données sont ici accessibles aussi grâce à leurs labels.

```
>>> s2[['a', 'c']] # Access values with labels 'a' and 'c'
```

DataFrame

Une DataFrame est une structure de données à 2 dimensions avec des colonnes labélisées, potentiellement de différents types. On peut voir un DataFrame comme une feuille de calcul, une table SQL, ou un dictionnaire (≈ une map) d'objets Series. Lorsque l'on manipule une DataFrame, les opérations sur les lignes et sur les colonnes sont traitées relativement de la même manière, la donnée étant structurée au sein de blocs à deux dimensions plutôt que grâce à des listes ou des dictionnaires. Il existe de nombreuses manières de créer un DataFrame. Une manière commune de le faire est à partir d'un dictionnaire de listes ou d'arrays NumPy.

```
>>> df = pd.DataFrame(np.random.randint(0, 10, (6,4)), columns=('A','B','C','D')) # Create a DataFrame from a NumPy array
>>> df2 = pd.DataFrame({'state': ['Ohio', 'Ohio', 'Nevada', 'Nevada'],
  'year': [2000, 2002, 2001, 2002],
  'pop': [1.5, 3.6, 2.4, 2.9]}) # Create a DataFrame from a dictionary
>>> df2.dtypes # Types of all columns
```

Il est aussi possible de créer un DataFrame directement en chargeant un fichier csv à l'aide de la méthode *pd.read_csv()*. Il faut alors éventuellement spécifier le séparateur et d'autres paramètres relatifs au fichier. D'autres formats de lecture sont aussi disponibles.
La méthode describe permet d'afficher un résumé des statistiques de vos données numériques (somme, moyenne, min, max, quartiles, etc.) :

```
>>> df.describe()
```

	A	**B**	**C**	**D**
count	6.000000	6.000000	6.000000	6.000000
mean	3.666667	3.500000	4.000000	4.333333
std	2.732520	3.082207	3.898718	2.804758
min	1.000000	0.000000	1.000000	0.000000
25%	2.000000	1.250000	1.250000	2.750000
50%	2.500000	3.000000	2.000000	5.000000
75%	5.250000	5.500000	7.250000	6.500000
max	8.000000	8.000000	9.000000	7.000000

De nombreuses méthodes permettent de récupérer l'index, les colonnes ou les données sous différentes formes

```
>>> print df.index # Return all indexes
>>> print df.columns # Return columns names
>>> print df.values # Values as a NumPy array
>>> df['A'] # Series corresponding to the column labeled 'A'
>>> df.A # Same as above
>>> df[['A','C']] # DataFrame corresponding to the columns labeled 'A' and 'C'
```

La sélection de données peut aussi se faire par label, position ou indexation booléenne.

```
>>> df.iloc[3] # Fourth line
>>> df.iloc[1,1] # Scalar element at line 2, column 2
>>> df[df.A > 5] # New DataFrame generated from boolean values
```

Les fonctions NumPy comme *min, max* ou *abs* fonctionnent aussi sur des DataFrame. Il est, de plus, possible d'utiliser la methode *apply* pour appliquer une fonction sur des arrays à une dimension. Prenons l'exemple suivant, qui calcule pour chaque colonne une valeur correspondant à la différence entre leur max et leur min. On peut pour

cela faire appel à des fonctions lambda, qui sont très utilisées en programmation fonctionnelle:

```
>>> f = lambda x: x.max() - x.min() # Lambda function subtracting max and min of a column
>>> df.apply(f, axis=0) # Apply f column-wise
```

Les DataFrames permettent d'explorer les données avec une syntaxe très simple et via de nombreuses méthodes utiles. Si vous possédez par exemple une colonne avec des données catégorielles, il est possible de compter le nombre d'occurrences de chaque catégorie grâce à la méthode *value_counts()*

```
>>> letters = pd.Series(['a', 'b', 'c', 'c', 'a', 'c', 'b', 'a', 'b', 'a', 'c', 'c', 'c', 'b', 'a'])
>>> letters.value_counts()
```

Lorsque des données sont manquantes, Pandas les remplace par des NaN (Not a Number). Il est alors toujours possible de faire des calculs en omettant les NaN. On peut filtrer les lignes avec des NaN en utilisant la méthode *dropna()*, ou bien remplacer les NaN par des valeurs (0 par exemple) en utilisant *fillna(0)*.
D'autres fonctionnalités sont disponibles avec Pandas, comme l'indexing hiérarchique, le groupement de deux DataFrame à l'aide de *groupby* ou *concat* ou encore l'utilisation de méthodes pour représenter graphiquement les données (Pandas se base sur la librairie matplotlib, qui est la librairie de graphe en Python la plus utilisée).

Le Machine Learning avec Scikit-learn

Créée en 2007, Scikit-learn est l'une des librairies open source les plus populaires pour le Machine Learning en Python. La librairie est construite à partir de NumPy, Matplotlib et SciPy.
En plus de contenir et de rendre accessible facilement tous les principaux algorithmes de Machine Learning (**classification**, **clustering**, **régression**), elle contient aussi de nombreux modules pour la **réduction de dimension**, l'**extraction de features**, le **processing des données** et l'**évaluation des modèles**.

Brève introduction au Machine Learning

Le but de cet article n'est pas de présenter le Machine Learning au lecteur, mais quelques notions de base sont indispensables afin de comprendre comment et dans quelles situations utiliser Scikit-learn. *Pour approfondir le sujet, je vous invite à parcourir nos articles de blog (blog.xebia.fr) sur le sujet.*
Le Machine Learning est la branche de l'Intelligence Artificielle qui s'occupe de la construction et de l'étude de systèmes qui apprennent à partir de données. On distingue deux principales catégories d'algorithmes d'apprentissage : supervisé et non-supervisé.
Dans l'apprentissage supervisé, on a en notre possession un dataset contenant à la fois des caractéristiques et des labels. On peut ainsi voir la variable label (notre cible) comme une fonction des caractéristiques. L'objectif est alors d'apprendre cette fonction afin de pouvoir prédire le label d'un nouvel objet étant donné ses caractéristiques. On peut par exemple vouloir prédire le prix d'une maison (le label est donc le prix) en fonction de sa surface, son nombre de chambres, etc. On parle ici de régression, car la variable à prédire est continue. On parle, au contraire, de classification lorsque l'on cherche à prédire une classe, comme par exemple le fait de prédire si un mail est un spam ou non à partir de son contenu.
En ce qui concerne l'apprentissage non supervisé, nous n'avons aucune information sur une éventuelle variable à prédire. Seul un dataset de caractéristiques est à notre disposition et l'objectif est alors d'extraire une structure des données. On parle notamment de clustering lorsque l'on cherche à diviser un ensemble de données en différents groupes homogènes, sans avoir d'information préalable sur la nature de ces groupes. La démarche en Machine Learning est relativement systématique, quelle que soit l'application : L'algorithme apprend un modèle sur un dataset d'apprentissage (qui a souvent eu besoin au préalable d'être traité), et est capable de faire des prédictions pour de nouvelles données. Ainsi, si nous avons un historique de maisons avec leurs caractéristiques ainsi que leur prix de vente, il est possible de construire un modèle sur cet historique, et de l'utiliser pour prédire le prix d'une nouvelle maison à estimer.
Maintenant que nous maîtrisons un peu plus de vocabulaire relatif au Machine Learning, nous pouvons commencer à utiliser Scikit-learn et à découvrir ses différentes fonctionnalités.

Scikit-learn : une excellente documentation

Que l'on débute dans la Data Science ou bien que l'on soit un utilisateur expérimenté, la documentation attachée à Scikit-learn est d'un recours très efficace. Toutes les méthodes sont bien documentées et l'on peut trouver de très nombreux exemples d'utilisation pour tous types de tâches. La documentation fournit même des conseils quant à l'utilisation de tel ou tel modèle de Machine Learning en fonction de vos besoins et de vos données disponibles, sous forme d'un schéma global facile à utiliser, visible Fig.1.

Une utilisation simple des algorithmes grâce à une API consistante

La plupart des algorithmes de Scikit-learn utilisent des jeux de données sous forme de tableaux (ou de matrices) à deux dimensions. Ces tableaux vont être soit du type numpy.ndarray, soit de type scipy.sparse

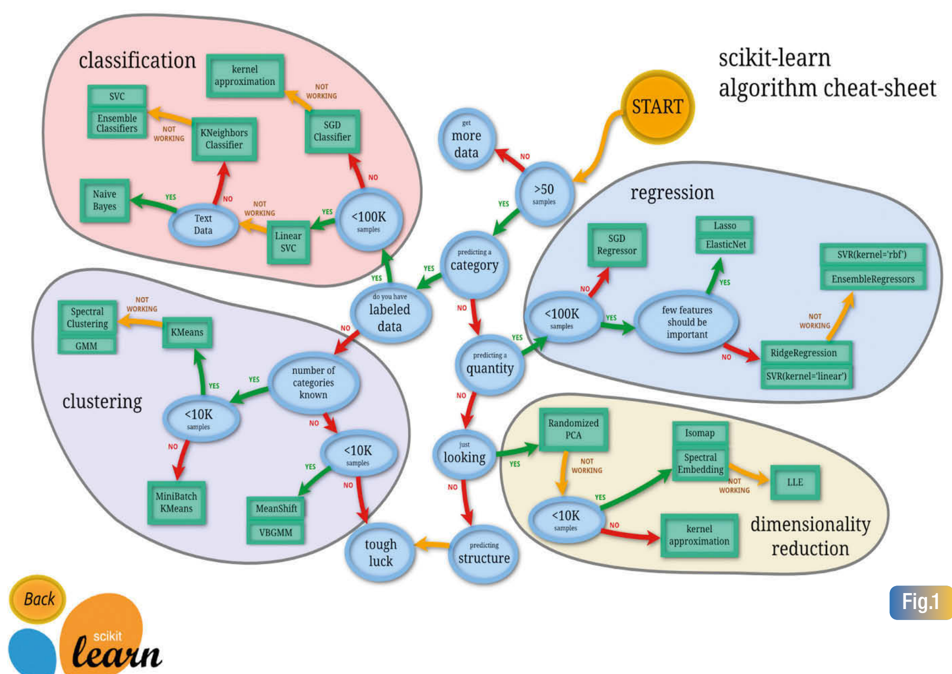


Fig.1

de la librairie SciPy. Il est aussi possible de fournir directement une DataFrame Pandas, la fonction prendra alors automatiquement son attribut *values* qui fournit les données sous forme d'array NumPy. La dimension de cette matrice sera [n_samples, n_features], avec:

- **n_samples:** le nombre d'échantillons de données ou d'observations. Un échantillon peut représenter un document, une image, un son, une vidéo, un objet, une ligne dans une base de données ou d'un fichier .csv ;
- **n_features:** le nombre de caractéristiques qui décrivent une observation de manière quantitative. Ces données doivent être des nombres (réels ou entiers).

Dans le cadre d'un apprentissage supervisé, les algorithmes nécessitent de plus qu'on leur fournisse le vecteur des labels associés à chaque donnée (un array NumPy à une dimension par exemple).
La plupart des opérations vont être réalisées à l'aide d'objets de type *estimator*. Un algorithme de Machine Learning correspond à une classe de type estimator :

```
>>> from sklearn.linear_model import LinearRegression
>>> model = LinearRegression(normalize=True)
>>> print model
```

scikit-learn s'efforce d'avoir une interface uniforme pour tous les algorithmes, ce qui est une de ses plus grandes forces. Étant donné un objet *estimator model*, plusieurs méthodes sont disponibles.
Pour tous les estimators :

- model.fit() : lance le mécanisme d'apprentissage.
  - model.fit(X, y) pour les algorithmes supervisés (X les observations d'entraînement, y leurs labels) ;
  - model.fit(X) pour les algorithmes non-supervisés.

Pour les estimators supervisés :

- model.predict(X_new) : prédit et retourne les labels d'un nouveau jeu de données X_new ;
- model.predict_proba(X_new) : pour les algorithmes de classification, prédit et retourne les probabilités d'appartenance à une classe d'un nouveau jeu de données X_new ;
- model.score(X_test, y_test) : retourne un score de performance du modèle entre 0 (mauvais) et 1 (idéal mais suspicieux).

Pour les estimators non-supervisés:

- model.transform(X_new) : transforme un jeu de données selon le modèle ;
- model.fit_transform(X) : apprend ses paramètres grâce à un jeu de données et transforme celui-ci.

Mise en situation

Nous allons maintenant mettre en application les notions introduites jusque là sur Pandas et Scikit-learn sur un exemple concret.
Scikit-learn permet de télécharger des jeux de données permettant de tester des modèles. Nous allons ici travailler sur le jeu de données *california housing data*, où l'objectif est de prédire le prix d'une maison à partir de caractéristiques (appelées features) telles que l'âge de la maison, la surface moyenne des chambres, la population dans la région ou encore les coordonnées géographiques.

```
>>> import pandas as pd
>>> from sklearn import datasets
>>> dataset = datasets.fetch_california_housing() # Downloads the dataset
>>> print dataset.feature_names
>>> dataset.data.shape
```

Comme on peut le constater, nous avons en notre possession des données concernant 20640 maisons, et 8 caractéristiques pour chaque maison. Le prix correspondant est stocké dans *dataset.target*. A noter que les données ont été modifiées pour ne pas correspondre aux valeurs réelles.
Avant de vouloir construire un modèle sur ces données, il est toujours important de les explorer et de comprendre leur sens. Pour cela, quoi de mieux que de les mettre sous forme d'une DataFrame Pandas et de faire appel à ses fonctionnalités.

```
>>> data_df = pd.DataFrame(data=dataset.data, columns=dataset.feature_names) # Create a Data Frame corresponding to the data and the columns names
>>> data_df.head() # Print the first lines of the dataset
>>> data_df.describe(percentiles=[0.25, 0.5, 0.75, 0.99]) # General statistics on all columns
>>> target_df = pd.DataFrame(data=dataset.target, columns=["Price"]) # DataFrame corresponding to the target
```

La méthode *describe()* permet notamment de détecter des outliers dans les données. Et comme on peut le constater, de nombreux sont présents dans ce jeu de données. Si l'on compare les valeurs entre le 99ème percentile et la valeur maximale de chaque variable, on constate parfois de très gros écarts (notamment avec AveOccup, AveRooms, AveBedrms). Ils sont d'autant plus visibles lorsque l'on dessine les distributions, chose facilement faisable en pandas avec la méthode *hist()*, qui se base sur matplotlib (Fig.2).

```
>>> %matplotlib inline # Allows to draw plots in the notebook
>>> data_df.hist(bins=50, figsize=(15,7)); # Histograms for all numerical columns of the DataFrame
```

Lorsque les distributions sont très clairement écrasées, cela signifie qu'il y a un ou plusieurs points très éloignés du reste. Ces outliers peuvent correspondre à des erreurs, ou bien peuvent représenter un sous-ensemble de la réalité qui est sous-représenté dans les données fournies. Il faut alors soit les remplacer par des valeurs adéquates, on parle alors d'imputation, soit supprimer les lignes en contenant, sans quoi nos modèles peuvent parfois être erronés s'ils sont sensibles aux outliers. Dans notre exemple, nous allons les supprimer. On ne va garder

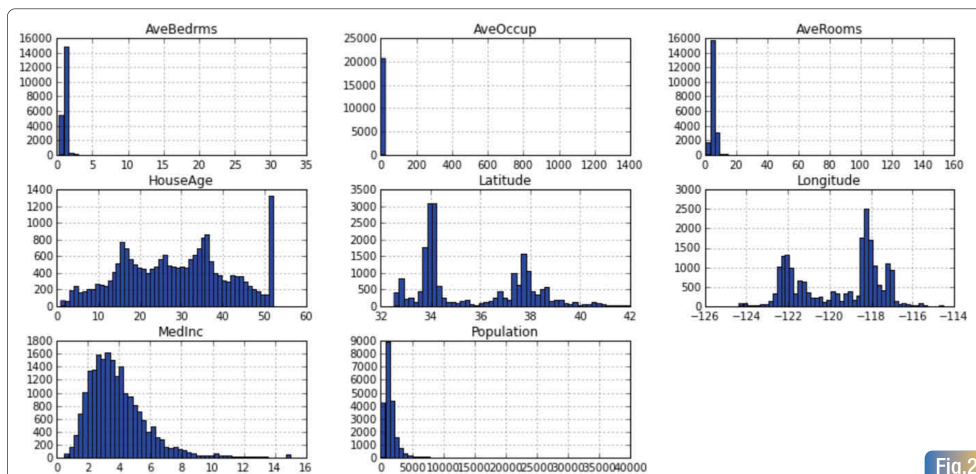


Fig.2

que les valeurs jusqu'au 99ème percentile. Ceci est très simple en Pandas, il suffit de lui fournir un vecteur de booléens contenant *False* là où il y a des outliers (Fig.3). Ne pas oublier de changer la target en conséquence.

```
>>> data_clean = data_df[((data_df.HouseAge<52) & (data_df.AveRooms<10.5) & (data_df.Ave
Bedrms<3) &
            (data_df.Population<6000) & (data_df.AveOccup<6) & (target_df.Price<5))].copy() # 
Removing outliers in DataFrame of values
>>> target_clean = target_df[((data_df.HouseAge<52) & (data_df.AveRooms<10.5) & (data_df.
AveBedrms<3) &
            (data_df.Population<6000) & (data_df.AveOccup<6) & (target_df.Price<5))].copy() # 
Removing outliers in DataFrame of target
>>> data_clean.hist(bins=50, figsize=(15,7)); # Histogram of the clean data
```

Maintenant que Pandas nous a permis de préparer nos données, on peut utiliser Scikit-learn pour construire un modèle dessus. Afin de tester les performances du modèle, il est important de séparer les données en un train et un test set. Le modèle va être appris sur le train set (qui représente la majorité des données). Le test set va être utilisé comme si on ne connaissait pas la target, le but sera alors de comparer la prédiction du modèle sur ce test set avec les vraies valeurs; cela permet d'évaluer le modèle, l'idée étant d'avoir une estimation de l'erreur. Cette séparation se fait très simplement avec Scikit-learn grâce à la méthode *train_test_split()*.

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(data_clean,target_clean, test_size=0.2)
# We take 20% of the data for final testing
```

Place maintenant à la construction de notre modèle de Machine Learning. Nous allons utiliser ici un modèle simple de Régression Linéaire, pour lequel le but est de trouver la meilleure combinaison possible des caractéristiques afin de minimiser la moyenne du carré des erreurs entre la prédiction et la vraie valeur pour chaque donnée d'apprentissage.
Cependant, comme on l'a vu précédemment, quel que soit l'algorithme choisi, l'utilisation de ce dernier se fait de manière identique, seuls les paramètres des modèles sont à changer.
L'utilisateur pourra alors tester d'autres modèles pour améliorer les performances que l'on obtiendra. Une fois le modèle chargé, il suffit de l'entraîner sur le train set à l'aide de la méthode *fit()* et de faire des prédictions sur le test set à l'aide de la méthode *predict()*.

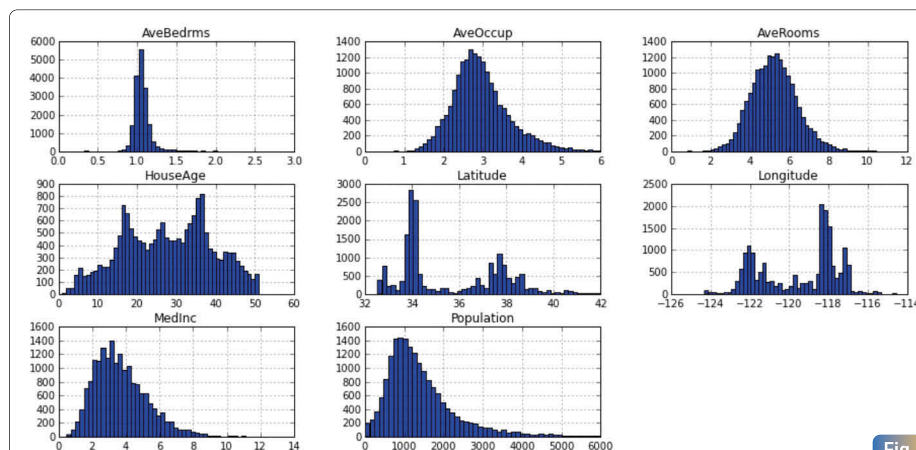


Fig.3

```
>>> from sklearn.linear_model import LinearRegression
>>> model = LinearRegression(fit_intercept=True, normalize=False) # Create a Linear Regression model
>>> model.fit(X_train, y_train) # Train the model on the train set
>>> pred = model.predict(X_test) # Predict on the test set
```

Il est important de normaliser les données (*normalize=True*), sans quoi les caractéristiques avec les plus grandes valeurs seront prépondérantes. Maintenant que nous avons des prédictions, il faut les comparer avec les vraies valeurs du test set.

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(pred, y_test)
```

On obtient un score de 0.33 (le but étant d'être proche de zéro), ce qui permet d'avoir une première idée des scores obtenables sur ce dataset. Il est à noter que le modèle en l'état est trop simple pour caractériser le phénomène étudié. Une approche possible pour y remédier serait d'ajouter des variables d'ordres supérieurs. Une seconde approche est d'envisager des modèles plus complexes (avec moins de biais) mais il faudra alors faire attention à ne pas overfitter (coller aux données d'apprentissage). En effet l'overfitting peut être vu comme un apprentissage par coeur et le modèle serait alors incapable de généraliser et d'avoir de bonnes performances sur des données encore jamais observées. L'utilisateur curieux pourra tenter d'utiliser d'autres algorithmes réputés plus performants, tels que les RandomForestRegressor, en faisant bien attention aux paramètres requis, en effet ce type de modèle requiert la spécification de certains paramètres, appelés “hyperparamètres”. On parle alors de tuning du modèle. scikit-learn dispose à cet effet de méthodes tel le *GridSearch* pour assister l'utilisateur dans la sélection du jeu de paramètres “optimal” pour le modèle, optimal au sens de la fonction à optimiser (ici le mean_squared_error) et relativement à la grille (discrète) spécifiés par l'utilisateur.

Il existe de très nombreuses autres fonctionnalités associées à la librairie Scikit-learn, on peut notamment citer la réduction de dimension (PCA).

Conclusion

De nombreuses autres librairies Python peuvent être utilisées pour travailler sur des données, on peut notamment citer **statsmodel** (modèles statistiques) ou **nltk** (traitement de données textuelles). Nous vous avons introduit ici les plus utilisées.
Python est un langage très couramment utilisé en Data Science, nous avons vu pourquoi grâce à ses différentes librairies. Il est relativement simple à prendre en main pour faire du travail sur la donnée, et tout est fait pour nous faciliter la tâche dans la réalisation d'opérations courantes qui peuvent être parfois assez complexes. Enfin, pour des problématiques de Big Data, le projet Spark (très en vogue en ce moment) propose une API en Python, appelée PySpark, montrant l'importance de ce langage dans le monde de la Data Science.

Références

Python for Data Analysis, O'Reilly
https://github.com/jakevdp/sklearn_pycon2014

Je pratique le C++

Partie 3/5.

Nous vous proposons une série d'articles sur la pratique de C++ pour que vous puissiez tous vous y mettre. Ce mois-ci on aborde les templates C++. Les templates représentent la partie programmation générique du langage C++.

Level 400



Christophe PICHAUD | .NET Rangers by Sogeti
Consultant sur les technologies Microsoft
christophepichaud@hotmail.com | www.windowscpp.net





La programmation orientée objet (OOP) et la programmation générique travaillent avec des types qui ne sont pas connus au moment où le programme est écrit. La distinction entre les deux est que OOP travaille avec des types qui ne sont connus qu'au runtime, tandis que la programmation générique utilise des types qui sont connus à la compilation.

Les containers, les itérateurs et les algorithmes sont des exemples de programmation générique. Quand on écrit un programme générique, le code est indépendant des types qu'il manipule. Quand on utilise un programme générique, nous fournissons les types ou les valeurs sur l'instance du programme qui va tourner. Par exemple, la librairie standard fournit une simple définition générique de chaque container, comme vector. Nous pouvons utiliser la définition générique pour définir plusieurs types de vector, chacun étant différent des autres, suivant le type qu'il contient. Les templates sont les fondations de la programmation générique en C++. Nous pouvons les utiliser sans comprendre comment ils sont définis. Un template est une « formule » pour créer des classes ou des fonctions. Lorsque nous utilisons un type générique comme vector, ou une fonction générique comme find, on fournit les informations de type nécessaire à l'exécution de cette classe ou fonction à la déclaration. Exemple : vector<string> v.

Concepts et programmation générique

Pourquoi les templates sont-ils fait ? Quelles techniques de programmation sont mises en œuvre quand on utilise les templates ? Les templates offrent :

- La possibilité de passer des types (des valeurs et des templates) en tant qu'arguments sans perte d'information.
- La vérification de type faite à l'instanciation.
- La possibilité de passer des valeurs constantes comme arguments. Cela implique de faire du calcul à la compilation.

Les templates fournissent un mécanisme puissant de compile-time computation et de manipulation de type qui permettent d'avoir un code efficace et très compact. Rappelons-nous que les types (classes) peuvent contenir du code et des valeurs. Le premier et le plus commun scenario d'utilisation des templates est le support de la programmation générique, qui est un modèle de programmation mettant l'accent sur le design, l'implémentation et l'utilisation d'algorithmes généraux. Ici, « général » veut dire qu'un algorithme peut être conçu pour accepter un large panel de types tant qu'ils sont passés en arguments. Les templates sont (compile-time) un mécanisme de polymorphisme paramétrique.

Considérez la fonction sum :

```
template<typename Container, typename Value>
Value sum(const Container& c, Value v)
{ for ( auto x : c ) v+=x;
return v;
}
```

Elle peut être invoquée pour n'importe quelle structure de données qui supporte begin() et end() de telle manière que le range-for puisse s'exécuter. De telles structures de la librairie standard (Standard Template Library) comme vector, list et map font l'affaire. Pour aller plus loin, le type d'élément de cette structure de données est seulement limité par son utilisation : elle doit être un type auquel on peut ajouter un argument Value. Les exemples sont ints, doubles et Matrices.

On peut dire que l'algorithme sum() est générique dans 2 dimensions : le type de data structure pour stocker les éléments (le container) et le type de ses éléments. Donc, sum() requiert que le premier argument soit une sorte de container, et le second argument de template soit une sorte de nombre. De tels prérequis se nomment des concepts.

De bons et précieux concepts sont fondamentaux pour la programmation générique. Les exemples sont les entiers et les nombres à virgules flottantes (comme définis même en C classique), et plus généralement des concepts mathématiques comme les vector et les containers. Ils représentent les concepts fondamentaux pour un champ d'applications. L'identification et la formalisation à un degré nécessaire pour une programmation générique efficace peut être un challenge.

Pour une utilisation basique, considérez le concept Régulier. Un type est régulier lorsqu'il se comporte comme un int ou un vector.

Un objet de type régulier :

- Peut être construit par défaut
- Peut être copié en utilisant un constructeur ou une affectation
- Peut être comparé en utilisant == et !=
- Ne souffre pas de problème technique à l'utilisation

Une string est un autre exemple de type régulier. Comme int, string est aussi Ordonné.

Cela veut dire que 2 strings peuvent être comparées avec <, <=, >, >= avec les sémantiques appropriées. Les concepts, ce n'est pas juste une notion syntaxique, c'est fondamentalement de la sémantique.

Les fonctions templates

Imaginons une fonction compare qui fonctionne avec tous les types, comment allons-nous écrire cela :

```
template <typename T>
int compare(const T &v1, const T &v2

{
   if (v1 < v2) return -1;
   if (v2 < v1) return 1;
   return 0;
}
```

Le return 0 est juste là pour faire joli car il faut finir la fonction et retourner quelque chose... C'est une simple fonction qui prend deux types T en paramètre. Remarquez l'élégance du style. Dans une définition de template, la liste des paramètres de template ne peut être vide.

Une définition de template commence avec le mot-clé template suivi d'une liste de paramètres de template qui sont séparés par des virgules et qui sont entre les token < et >.

Instanciation d'une fonction template

Quand on appelle une fonction template, le compilateur utilise les arguments de l'appel pour déduire les arguments de template pour nous. Lors

de l'appel à compare, le compilateur utilise le type des arguments pour déterminer quel type associer au paramètre de template T.

```
cout << compare(1, 0) << endl;      // T est int
vector<int> vec1{1, 2, 3}, vec2{4, 5, 6};
cout << compare(vec1, vec2) << endl; // T est vector<int>
```

Pour le premier appel, le compilateur va écrire et compiler une version de compare avec T remplacé par int :

```
int compare(const int &v1, const int &v2)
{
   if (v1 < v2) return -1;
   if (v2 < v1) return 1;
   return 0;
}
```

Pour le second appel, la fonction sera générée avec des vector<int>.

Les paramètres de type template

Il est possible d'utiliser le paramètre de type template comme les arguments de la fonction. Exemple :

```
template <typename T> T foo(T* p)
{
  T tmp = *p;
   // ...
   return tmp;
}
```

Chaque paramètre de type doit être précédé du mot clé class ou typename.

```
template <typename T, U> T calc(const T&, const U&);
template <typename T, class U> calc (const T&, const U&);
```

La compilation des templates

Quand le compilateur voit la définition d'un template, il ne génère pas de code. Il génère du code seulement quand on instancie une instance spécifique d'un template.

Le fait que le code soit généré seulement quand on utilise le template (et non quand on le définit) affecte la manière avec laquelle nous organisons le source code et la manière avec laquelle les erreurs sont détectées. Il en va aussi de la taille de l'exe ou de la librairie lib ou dll.

Quand on appelle une fonction, le compilateur a besoin de voir seulement la déclaration de la fonction. Quand on utilise un objet de type de class, la définition de classe doit être disponible mais la définition des fonctions membres n'a pas besoin d'être présente.

Donc, on met les définitions de classes et les déclarations de fonctions dans des fichiers d'entêtes (.h), et les définitions des fonctions membres de classes dans le source code (.cpp).

Les templates sont différents. Pour générer une instanciation, le compilateur a besoin d'avoir le code qui définit une fonction template ou une fonction membre de classe template.

Donc, les entêtes (.h) pour les templates contiennent leurs définitions et leurs déclarations et leurs fonctions membres.

Les erreurs de compilation des templates

La détection des erreurs sur les template peut être parfois un véritable parcours du combattant car le compilateur vérifie plusieurs choses lors de l'instanciation du template. Une des erreurs les plus fréquentes est celle rencontrée sur les types. Par exemple : un template requiert une opération cout <<. La compilation ne détecte pas d'erreur tant que l'instanciation sur un type n'est pas effectuée. Si le type ne supporte pas l'opérateur << pour iostream cout sur l'objet passé dans le paramètre, l'erreur est immédiatement détectée.

Exemple :

```
template <typename T> void Echo(const T & t)
{
            cout << t << endl;
}

void Template1()
{
            string t = "My Lisa";
            Echo(t); // OK
            Echo(9); // OK

            vector<string> girls = { "edith", "lisa", "audrey" };
            Echo(girls); // KO, binary << no operator found in vector<T>...
}
```

Erreur de compilation : binary '<<': no operator found which takes a right-hand operand of type 'const std::vector<std::string,std::allocator<_Ty>>' (or there is no acceptable conversion)

Dans notre exemple, il n'existe aucun opérateur << défini dans vector<T> qui puisse travailler avec la fonction cout définie dans iostream de la STL. La fonction Echo ne fait que un cout << mais ce n'est pas possible sur vector<T>. C'était possible sur string et sur int mais pas sur vector<T>.

Une bonne pratique consiste à essayer de minimiser le nombre de prérequis demandés sur le type d'argument.

C'est au designer du template de vérifier que les arguments passés au template supportent toutes les opérations que le template utilise et que ces opérations se comportent correctement dans le contexte que le template utilise.

Les templates de classes

Un template de classe sert à générer des classes. Les templates de classes diffèrent des templates de fonctions dans le fait que le compilateur ne peut pas déduire les types de paramètres du template pour un template de classe. Pour utiliser un template de classes, nous devons fournir une information additionnelle dans les token (< et >) juste après le nom du template. Les informations en extra sont la liste des arguments du template à utiliser pour ce template.

Définir un template de classe

Comme les templates de fonctions, les templates de classes commencent par le mot-clé template suivi d'une liste de paramètres du template. Voici un template pour la gestion des pointeurs. Plus besoin de faire de delete, le template s'en charge.

```
template<typename T>
class MyPtr
{
public:
   MyPtr()
   {
```

```
        m_count = 0;
        m_ptr = nullptr;
        cout << "No Pointer catched" << endl;
    }

    MyPtr(T* ptr) : m_ptr(ptr)
    {
        m_count = 0;
        m_count++;
        cout << "Pointer catched" << endl;
    }

    virtual ~MyPtr()
    {
        m_count--;
        if (m_count == 0)
        {
            delete m_ptr;
            cout << "Pointer deleted !" << endl;
        }
    }

    T& operator*(void)
    {
        return *m_ptr;
    }

    T* operator->(void)
    {
        return m_ptr;
    }

    MyPtr& operator=(MyPtr<T> &ptr)
    {
        if ( m_ptr != nullptr)
            delete m_ptr;
        m_ptr = ptr.m_ptr;
        m_count++;
        return *this;
    }

    MyPtr& operator=(T* ptr)
    {
        if (m_ptr != nullptr)
            delete m_ptr;
        m_ptr = ptr;
        m_count++;
        return *this;
    }

private:
    T * m_ptr;
    int m_count;
};
```

Le template MyPtr possède un type de paramètre template nommé T. Nous pouvons utiliser ce paramètre n'importe où pour représenter le type que MyPtr détient. Par exemple, nous définissons le type de retour d'une opération qui fournit accès à l'élément de MyPtr comme T&.

Quand un développeur va instancier ce template, les utilisations de T seront remplacées par un argument de type template spécifique.

Instanciation du template de classe

Pour instancier le template de classe, il faut fournir explicitement un type de paramètre au template comme, par exemple, une classe CElement qui est une classe fictive pour désigner des éléments à dessiner:

```
MyPtr<CElement> pElement(new CElement(10));
pElement->Draw();
```

Il est possible d'avoir en données membre un stockage des éléments dans vector par exemple en utilisant le paramètre de type du template :

```
std::shared_ptr<std::vector<T>> data;
```

Le type T est type comme les autres donc on peut l'utiliser comme on veut, en T, en T& en T*. Toutes les combinaisons sont possibles.

Les fonctions membres des templates de classe

Les fonctions membres peuvent être définies dans le header ou dans le corps. Si elle le sont hors du header, il faut spécifier template<T> et le nom de la classe suivi de :: et du nom de la méthode.
Exemple :
Dans le header :

```
  MyPtr& operator=(MyPtr<T> &ptr);
```

Dans le cpp :

```
template<typename T>
MyPtr<T>& MyPtr<T>::operator=(MyPtr<T> &ptr)
{
    if (m_ptr != nullptr)
        delete m_ptr;
    m_ptr = ptr.m_ptr;
    m_count++;
    return *this;
}
```

Dans le template MyPtr<T>, on distingue plusieurs subtilités. Les opérateurs * et -> ont été redéfinis pour une utilisation transparente du mécanisme des smart pointeurs (pointeurs intelligents).

Membres static et les templates

Il est possible de mettre des membres static dans les templates de classes.

```
template <typename T> class Foo {
public:
  static std::size_t count() { return ctr; }
private:
  static std::size_t ctr;
};
```

Tous les objets de type Foo<T> partagent les mêmes données static.

```
// instantiation des membres static Foo<string>::ctr and Foo<string>::count
Foo<string> fs;
```

```
// tous les 3 objets partagent les membres Foo<int>::ctr and Foo<int>::count
Foo<int> fi, fi2, fi3;
```

Templates et arguments par défaut

Il est possible de passer des types par défaut pour les paramètres de type d'un template.

```
template <typename T, typename F = less<T>>
int compare(const T &v1, const T &v2, F f = F())
{
  if (f(v1, v2)) return -1;
  if (f(v2, v1)) return 1;
  return 0;
}
```

Ici le deuxième paramètre du template est par défaut et c'est less<T>. La fonction less de la STL prend deux paramètres (x et y) et retourne x<y.

Spécialisation de template

Dans certains cas précis, on est obligé de demander un fonctionnement particulier du template pour un type de paramètre précis. Prenons un exemple :

```
template <typename T>
struct Wrap
{
    typename typedef T type;
    static const T& MakeWrap(const T& t)
    {
        return t;
    }
    static const T& Unwrap(const T& t)
    {
        return t;
    }
};
```

Ce template Wrap<T> représente un wrapper universel. Par défaut MakeWrap et Unwrap retourne T et le typedef type aussi retourne un T. Pour l'ensemble des types simples, ce wrapper fonctionne mais pour les type HSTRING et les pointers, il faut wrapper cela autrement. Ce développement est spécifique Windows 8.

```
template <>
struct Wrap<HSTRING>
{
    typedef HStringHelper type;
    static HStringHelper MakeWrap(const HSTRING& t)
    {
        HStringHelper str;
        str.Set(t);
        return str;
    }
    static HSTRING Unwrap(const HStringHelper& t)
    {
        HSTRING str;
        t.CopyTo(&str);
        return str;
    }
};

template <typename T>
struct Wrap<T*>
{
    typename typedef ComPtr<T> type;
    static ComPtr<T> MakeWrap(T* t)
    {
        ComPtr<T> ptr;
        ptr.Attach(t);
        return ptr;
    }
    static T* Unwrap(ComPtr<T> t)
    {
        ComPtr<T> ptr(t);
        return t.Detach();
    }
};
```

Le type HSTRING ne peut pas être manipulé tel-que car c'est comme un HANDLE. C'est la raison pour laquelle HSTRING est traité avec une classe HStringHelper et type devient HStringHelper Il en va de même pour les pointeurs qui doivent être encapsulés dans des ComPtr<T> car ce sont des composants COM. Le type est ComPtr<T> car les pointeurs sont des interfaces COM et il faut les encapsuler pour les utiliser. Oui c'est du développement Windows mais l'essentiel c'est de remarquer la spécialisation du template Wrap<T> et ses variantes Wrap<HSTRING> et Wrap<T*>.
Le concept clé c'est que les règles standard s'appliquent aux spécialisations. Pour spécialiser un template, une déclaration du template original doit être accessible. Il en va de même pour le template spécialisé avant son utilisation.
Avec des classes ou des fonctions ordinaires, les déclarations manquantes sont faciles à trouver. Pour les templates, c'est souvent plus compliquer. Surtout si la spécialisation du template n'est pas contenue dans le même fichier d'entête que le template principal. Les templates et les templates spécialisés doivent être déclarés dans le même fichier d'entête. En premier lieu, on définit le template original et ensuite les templates spécialisés.

Facilités de déclaration dans les templates

Dans certains cas, le typedef sur un type va permettre une meilleure lisibilité du code. Exemple :

```
template <class T>
class Iterator : public RuntimeClass<IIterator<T>>
{
    InspectableClass(L"Library1.Iterator", BaseTrust)

private:
    typedef typename std::vector<typename Wrap<T>::type> WrappedVector;
    typedef typename std::shared_ptr<WrappedVector> V;
    typedef typename WrappedVector::iterator IT;

 ../..
private:
    V _v;
    IT _it;
    T _element;
    boolean _bElement = FALSE;
};
```

Ce template Iterator<T> contient un WrappedVector qui est un typedef sur vector<typename Wrap<T> ::type>.
Le fait d'utiliser Wrap<T>::type fait la magie de ce template. Pour des types simples, T vaut T et pour HSTRING et T*, ça vaudra HStringHelper et ComPtr<T>. Ce mécanisme est très puissant.
Dans cet exemple, le template Iterator<T> représente un itérateur sous Windows 8 en mode Windows Store. Oui je sais c'est du Windows mais c'est le style qui compte. Regardez les typedef, ils utilisent Wrap<T> dans un vector<T> et l'itérateur du vector est défini en IT, plus simple à écrire.
La déclaration de V est un shared_ptr de vector<T> avec T qui vaut Wrap<T>. C'est assez complexe mais cela veut dire que c'est un container générique d'objets wrappés qui peut contenir à la fois des types simples mais aussi des HSTRING et aussi des pointeurs d'interfaces T* qui sera géré comme des pointeurs d'interfaces COM donc encapsulé avec la spécialisation de Wrap<T*> via ComPtr<T>.
Regardons comment est définie la méthode GetCurrent pour cette classe itérateur.

```
virtual HRESULT STDMETHODCALLTYPE get_Current(T *current)
{
    try {
        _LogInfo(L"Iterator::get_Current()...");
    if (_it != _v->end())
    {
        _bElement = TRUE;
        _element = Wrap<T>::Unwrap(*_it);
        *current = _element;
    }
    else
    {
        _bElement = FALSE;
    }
    return S_OK;
    _EXCEPTION_HANDLER(L"Iterator::get_Current()...");
}
```

GetCurrent doit retourner l'élément courant ; on vérifie l'itérateur pour savoir si on est à la fin ou pas. On Unwrap l'élément et on le retourne dans le paramètre de la fonction. Le booléen _bElement est positionné pour avoir une information en double sur le statut de position de l'itérateur. Il est utile pour une autre méthode du template pour savoir s'il existe un élément courant.
Vous allez me dire, OK on déclare un itérateur mais où est la classe du container qui utilise cet itérateur. Voici la classe Vector<T> spécifique pour Windows 8 qui permet de gérer un container générique en prenant soin des types simples et des types complexes (HSTRING, pointeurs d'interface COM) avec Wrap<T>.

```
template <typename T>
    class Vector : public RuntimeClass<IVector<T>,
        IIterable<T>,
        IObservableVector<T>>
    {
        InspectableClass(L"Library1.Vector", BaseTrust)

    private:
        typedef typename std::vector<typename Wrap<T>::type> WrappedVector;
        typedef typename WrappedVector::const_iterator CIT;
        typedef typename VectorChangedEventHandler<T> WFC_Handler;

    public:
        Vector()
        {
            _LogInfo(L"Vector::Vector()...");
            _v = std::make_shared<WrappedVector>();
            m_observed = false;
        }
    public:
        virtual HRESULT STDMETHODCALLTYPE GetAt(unsigned index, T *item)
        {
            _LogInfo(L"Vector::GetAt()...");
            *item = Wrap<T>::Unwrap((*_v)[index]);
            return S_OK;
        }

        ../..
        virtual HRESULT STDMETHODCALLTYPE First(IIterator<T> **first)
        {
            _LogInfo(L"Vector::First()...");
            ComPtr<Iterator<T>> p = Make<Iterator<T>>();
            p->Init(_v);
            *first = p.Detach();
            return S_OK;
        }
        ../..
private:
        std::shared_ptr<WrappedVector> _v;
        bool m_observed;
        EventSource<VectorChangedEventHandler<T>> m_events;
    };
```

Pour simplifier la compréhension de ce template Vector<T>, je n'ai fait figurer que la méthode GetAt qui permet de récupérer un élément et la méthode First qui retourne un itérateur sur le Vector<T> via Iterator<T>, template que nous avons découvert précédemment. Le template Vector<T> hérite de plusieurs classes qui sont aussi des templates.

Conclusion

Les templates sont des classes ou des fonctions qui permettent de générer des classes. Leur utilisation nécessite un peu de pratique mais c'est quelque chose d'appréhensible avec un peu d'effort. La puissance des templates réside dans la possibilité de définir une classe générique et de prévoir les adaptations nécessaires pour qu'elle puisse être utilisée avec le minimum de contraintes. La spécialisation partielle des templates est une formidable mécanique pour corriger les types un peu limités ou trop pénibles à gérer. Dans ces opérations, le typedef est ton ami. On déclare un T et puis avec son utilisation on se rend compte que T est trop simple et qu'il faut wrapper le T dans certains cas. C'est cool. Cet article vous permet de découvrir ce que sont les templates. Cela peut paraître compliqué mais avec un peu de pratique, on est complètement aspiré et tout devient limpide. Il faut toujours se mettre en situation avec la casquette du designer du template et changer de temps en temps en tant qu'utilisateur du template. La découverte des erreurs de compilations les plus complexes se fera via le code d'utilisation des templates. Enjoy ! Le code source Windows 8 autour de Vector<T>, Iterator<T>, IIterable<T> et Wrap<T> est accessible ici : http://code.msdn.microsoft.com/windowsapps/Windows-Runtime-Component-4dc6fa20



GodMode : le puissant outil caché de Windows 10

Le GodMode est une option cachée qui permet d'accéder à tous les paramètres de Windows 10 dans un seul endroit. Pour l'activer, créer un dossier à l'endroit de votre choix, dans notre exemple on va créer le dossier sur le bureau.


Azzam ALNIJRES
MVP (Windows Experience)
Membre de Microsoft STEP MCTS
Président Club Windows 10
windows10.club-windows.com

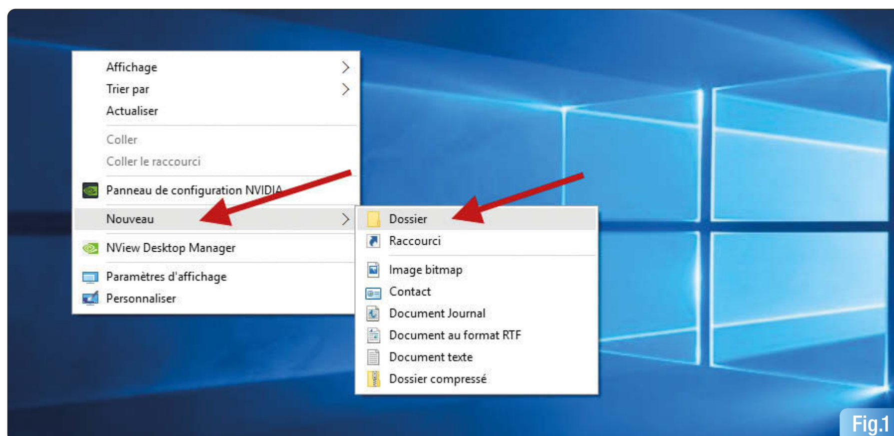


Fig.1

Un clic droit sur le bureau, puis Nouveau et Dossier

Renommez le dossier Fig.1 GodMode.{ED7BA470-8E54-465E-825C-99712043E01C}

Puis appuyez sur la touche Entrée pour valider Fig.2.

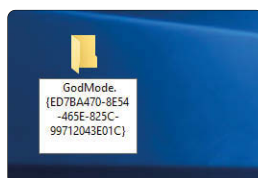


Fig.2

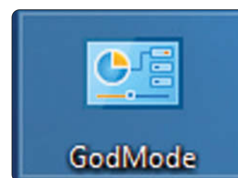
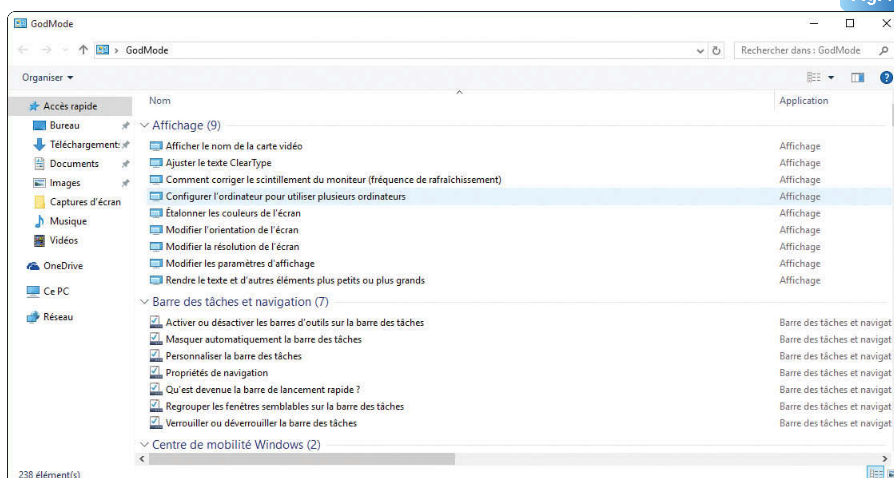


Fig.3

Windows affiche l'icône GodMode Fig.3.

Quand vous double cliquez sur GodMode, Windows vous affiche plus de 260 fonctions et outils Fig.4.

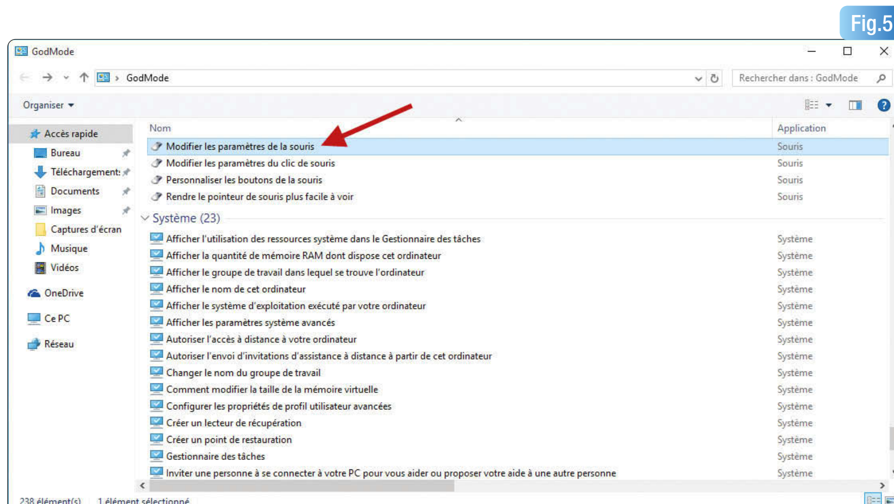
Fig.4



Quand vous cliquez sur une fonction, par exemple Modifier les paramètres de la souris Fig.5.

Les propriétés de Souris s'ouvrent Fig.6.

Fig.5



Les fonctions et outils sont classés dans les rubriques :

- Affichage
- Barre des tâches et menu Démarrer
- Centre de maintenance
- Centre de synchronisation
- Centre Réseau et partage
- Chiffrement de lecteur BitLocker
- Clavier
- Comptes d'utilisateurs
- Connexions distantes
- Contrôle parental
- Date et heure
- Emplacement et autres capteurs
- Exécution automatique
- Gadgets du Bureau
- Gestion des couleurs
- Gestionnaire de périphériques
- Gestionnaire d'identification
- Groupe résidentiel

- Icônes de la zone de notification
- Informations et outils de performance
- Mise en route
- Options d'alimentation
- Options d'ergonomie
- Options d'indexation
- Options des dossiers
- Options Internet
- Outils d'administration
- Pare-feu Windows
- Périphériques et imprimantes
- Personnalisation
- Polices
- Programmes et fonctionnalités
- Programmes par défaut
- Reconnaissance vocale
- Région et langue
- Résolution des problèmes
- Sauvegarder et restaurer
- Son
- Souris
- Système
- Téléphone et modem
- Windows CardSpace
- Windows Defender
- Windows Update

Vous pouvez également faire glisser l'une des commandes à votre bureau pour créer un raccourci pour la commande Fig.7.

Vous pouvez aussi épingler GodMode dans Accès rapide Fig.7 et 8.

Ou tout simplement l'épingler à l'écran de démarrage Fig.9 et 10.

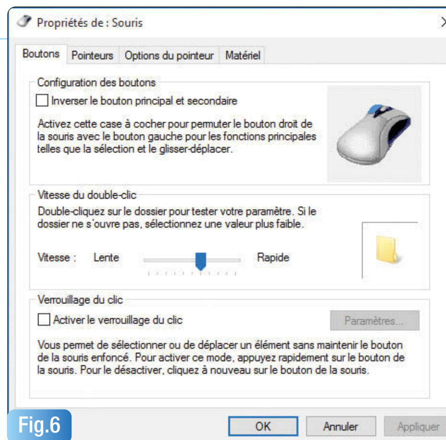


Fig.6

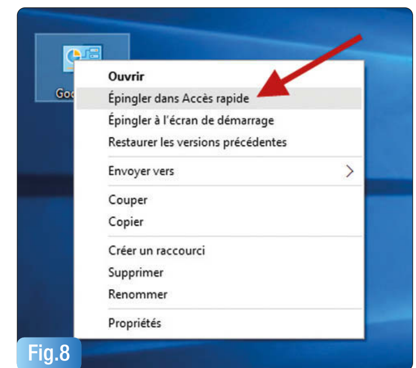


Fig.8

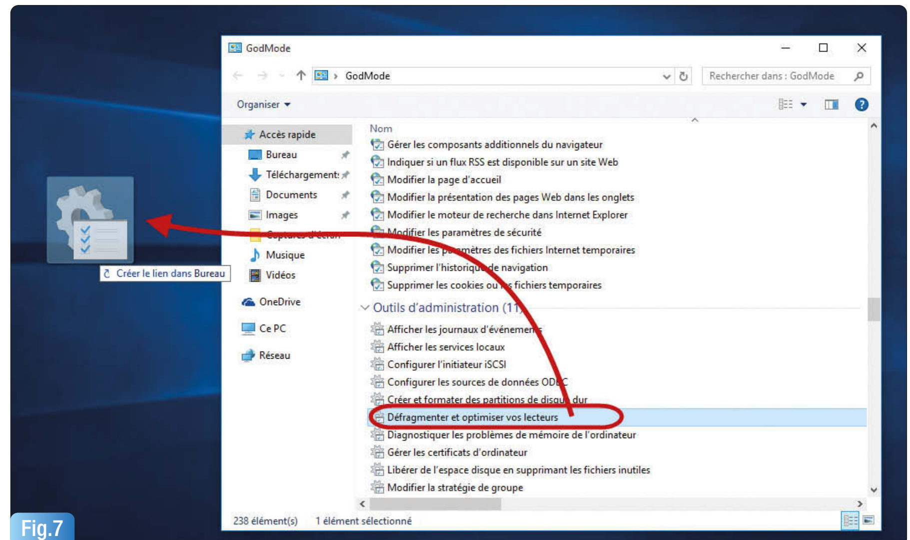


Fig.7

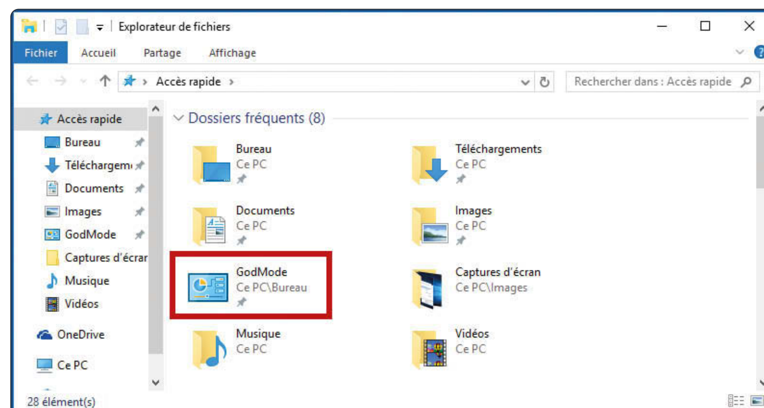


Fig.9

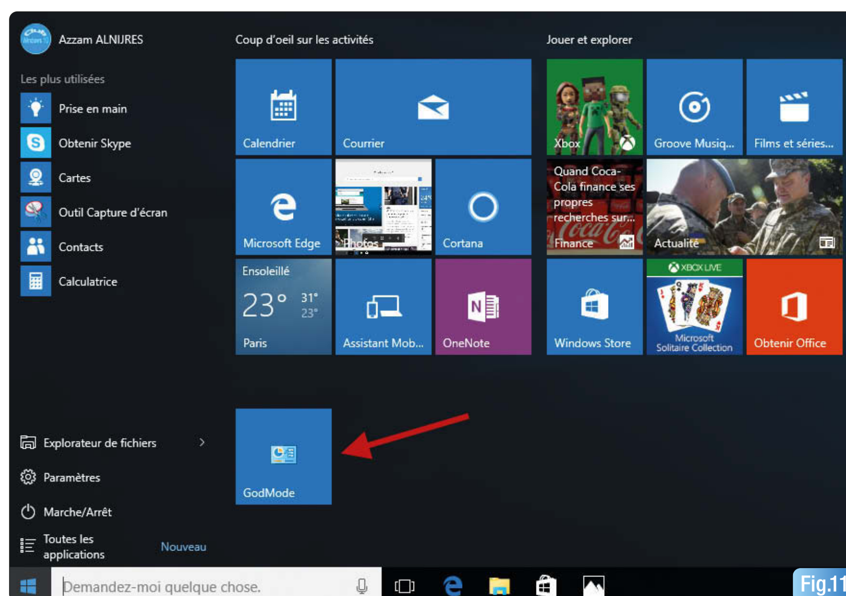
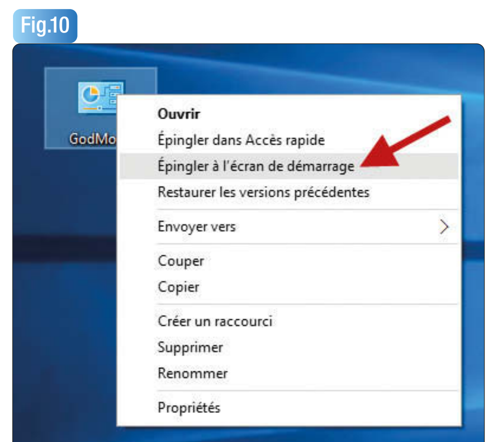


Fig.11

Fig.10


Node.js de A à Z

1ère partie

Je ne vous apprends rien en vous disant que beaucoup de développeurs Web utilisent JavaScript pour créer des applications front. Node.js permet à ce langage très populaire d'être utilisé dans plusieurs autres contextes, par exemple sur un serveur Web. Il existe plusieurs fonctionnalités notables offertes par Node.js ce qui le rend sans nul doute digne d'un grand intérêt.



Wassim Chegham (@manekinekko)
Expert en Nouvelles Technologies Web chez Groupe SII
Google Developer Expert (GDE) en Technologies Web

Node (pour les intimes) est un "wrapper" autour du célèbre runtime JavaScript V8 créé par Google et alimentant Google Chrome. Node apporte plusieurs optimisations à V8 lui permettant de fonctionner en dehors du navigateur. Par exemple, dans le cas d'un serveur Web, la manipulation des fichiers binaires est souvent nécessaire. Node enrichit V8, et donc JavaScript, avec une API de manipulation de données binaires (Buffer API).

Grâce à V8 et aux techniques récentes issues du domaine de la compilation (JIT, Concurrent Speculative Optimization...), JavaScript qui est un langage haut niveau à la base, est capable de produire des résultats de performances très proches de ceux produits par un langage bas niveau, comme le C par exemple. Voici un article [1] détaillant la prouesse de Mozilla dans ce domaine. En plus de ces aspects de performance proposés par V8, Node tire avantage de la nature même du langage JavaScript, à savoir l'aspect événementiel ; lui permettant ainsi de produire des serveurs très évolutifs et performants. Grâce à une architecture appelée la "boucle d'événements", Node rend la programmation côté serveur très simple. Programmer une tâche concurrente est un jeu d'enfant maintenant avec Node.

Pour supporter cette approche évènementielle, Node fournit une bibliothèque de librairies (ou interfaces) "non bloquantes", pour l'accès au système de fichiers ou à une base de données, par exemple. Lorsque vous soumettez une requête pour lire un fichier, au lieu de forcer Node à attendre que le disque se réveille et se positionne sur le fichier en question, l'interface "non-bloquante" notifie Node lorsque le fichier a été lu. De la même manière quand le navigateur notifie votre code à propos d'un événement souris ou clavier, ou la réponse d'une requête serveur (XHR).

Même si ce n'est pas vraiment lié à Node (comparatif des technologies JavaScript côté serveur), le fait de reposer sur JavaScript pour la programmation côté serveur reste un choix judicieux. Qu'on le veuille ou non, les navigateurs ne nous proposent qu'un choix très restreint de langages côté front. Si nous souhaitons donc partager du code entre le serveur et le client, JavaScript reste le seul et unique choix. Ceci est d'autant plus vrai maintenant, avec l'apparition des SPA (Single Page Application). Parce que nous devons compter sur JavaScript pour créer ces applications côté navigateur, le fait d'avoir un environnement JavaScript côté serveur rend le partage de code possible (validation de formulaire par exemple), ce qui est très compliqué à réaliser, voire impossible, avec d'autres technologies serveur comme PHP, Ruby, Python ou Java. De plus, JSON est un format d'échange très populaire de nos jours et JavaScript apporte un support natif de ce format. Enfin, plusieurs technologies de base de données NoSQL telles que CouchDB ou MongoDB sont basées sur JavaScript. Un seul langage pour les dominer tous...

Nous allons découvrir ensemble cette nouvelle plateforme; comprendre comment Node fonctionne en expliquant le paradigme de programmation évènementielle, très cher à JavaScript. Ensuite, nous verrons comment installer et configurer notre environnement Node. Nous introduirons brièvement ensuite NPM, l'outil de gestion des packages Node.

Après, nous introduirons quelques API et concepts proposés par Node, tels que les "streams". Nous expliquerons bien évidement comment développer une application Node, comment la tester et la déboguer. Nous expliquerons également la notion de module — permettant de composer une application — en téléchargeant des modules tiers et en créant nos propres modules. Ensuite, nous verrons quelques exemples d'accès à des bases de données, MongoDB et MySQL.

En bonus, nous expliquerons comment ajouter une application Node dans un processus d'intégration continue et découvrirons quelques astuces pour la déployer en production. Enfin, nous verrons comment nous pouvons assurer une "scalabilité" (ou mise à l'échelle) horizontale de notre application Node. Démarrons sans plus tarder notre aventure dans le monde magique de Node... Quelques prérequis tout de même avant de commencer cette série d'articles : je vais supposer que vous maîtrisez déjà JavaScript [1]. Je supposerai aussi que vous avez quelques notions en système UNIX, en TCP et HTTP et bien évidemment que vous avez déjà développé des applications Web.

Installation et configuration de Node

En ce qui concerne l'installation de Node, vous avez la possibilité soit de récupérer les binaires depuis le site officiel, soit de passer par NVM qui est un outil de gestion de versions pour Node. Je vous recommande d'utiliser NVM...

Installation de Node depuis le site officiel

Je pense que je n'ai pas besoin de trop m'attarder sur cette étape. Il vous suffit d'aller à cette adresse https://nodejs.org/download/ et de suivre les instructions pour récupérer la version adaptée à votre système.

Gestion des versions de Node avec NVM

Il peut arriver parfois qu'il faille jongler entre différentes versions de Node pour une raison X ou Y. NVM ou Node Version Manager est un outil inspiré de RVM (Ruby Version Manager) permettant d'installer, de gérer et configurer automatiquement plusieurs versions de Node sur une même machine. Pour récupérer NVM, lancer votre terminal et exécuter la commande suivante :

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.25.4/install.sh | bash
$ # Ou bien...
$ wget -q0- https://raw.githubusercontent.com/creationix/nvm/v0.25.4/install.sh | bash
```

Vous pouvez utiliser CURL ou WGET pour récupérer le script d'installation de NVM puis l'exécuter. Fig 1

Une fois terminé, le script vous invite à rouvrir votre terminal pour que l'installation prenne effet : Fig 2

Testez que NVM a bien été installé : Fig 3

Maintenant, il vous suffit d'installer une version de Node. Installons la dernière version... Fig 4

Je vous laisse découvrir NVM à tête reposée [2], et je vous conseille vivement de l'ajouter à votre stack d'outils.

Le mode REPL

Node offre un mode interactif pour tester rapidement quelques petits bouts de code. Pour lancer le REPL de Node, il suffit d'exécuter la commande **node** dans votre terminal : Fig 5

À noter que pour expérimenter les nouveautés d'ES6 (appelée aussi ES2015), vous devez lancer Node avec l'option **–harmony** (Harmony étant l'ancien nom de code du premier brouillon des spécifications d'ES6).

Exécution de script

Le mode est très pratique, mais pour développer et architecturer nos applications Node, nous allons écrire notre code dans des scripts JavaScript, puis utiliser node pour les interpréter : Fig 6

N'oubliez pas d'ajouter l'option **–harmony** !

Maintenant, que votre environnement est prêt ! Passons à l'étape suivante qui est la découverte de NPM, un autre compagnon de Node.

Découverte de NPM

Maintenant que Node est installé, vous allez pouvoir bénéficier de toutes les fonctionnalités offertes par Node et JavaScript. En effet, Node vient avec plusieurs modules "built-in" très puissants que nous découvrirons ultérieurement. Cependant, la plupart de ces modules proposent des API bas niveau plus ou moins simples à manipuler. Inclure des modules tiers devient nécessaire pour développer n'importe quelle application dite "complexe". Qui voudrait s'amuser à réinventer la roue ? Node vient packagé (depuis la version 0.6) avec un outil pour télécharger, installer et gérer des modules ou "packages" tiers développés par la communauté. Cet outil est le Node Package Manager, ou NPM pour les intimes. Lorsque nous parlons de NPM, généralement nous parlons de trois choses :

- Un service de dépôt public où sont référencés tous les modules tiers développés par la communauté. La quasi-totalité de ces modules est hébergée sur github.com.
- Un programme en ligne de commande pour télécharger, installer et gérer ces modules.

```
2. wchegham@wchegham-mbp-2: /tmp (zsh)
wchegham 192.168.0.30  /tmp
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.25.4/install.sh | bash
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  7149  100  7149    0     0  17585      0 --:--:-- --:--:-- --:--:-- 17608
=> Downloading nvm from git to '/Users/wchegham/.nvm'
=> Cloning into '/Users/wchegham/.nvm'...
remote: Counting objects: 3693, done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 3693 (delta 1), reused 0 (delta 0), pack-reused 3684
Receiving objects: 100% (3693/3693), 831.03 KiB | 511.00 KiB/s, done.
Resolving deltas: 100% (2126/2126), done.
Checking connectivity... done.
* (detached from v0.25.4)
  master
```

Fig.1

```
2. wchegham@wchegham-mbp-2: /tmp (zsh)

=> If you wish to uninstall them at a later point (or re-install them under your
=> `nvm` Nodes), you can remove them from the system Node as follows:

     $ nvm use system
     $ npm uninstall -g a_module

=> Close and reopen your terminal to start using nvm
wchegham 192.168.0.30  /tmp
$
```

Fig.2

```
2. wchegham@wchegham-mbp-2: /tmp (zsh)
wchegham 192.168.0.30  /tmp
$ nvm --version
0.25.4
wchegham 192.168.0.30  /tmp
$
```

Fig.3

- Un formalisme pour la définition ainsi que la gestion des dépendances entre les différents modules qui composent votre application. Ce formalisme est décrit dans un fichier nommé **package.json**.

Vous n'avez pas besoin de tout savoir sur NPM pour démarrer avec Node, mais tôt ou tard vous allez avoir besoin d'utiliser un programme en CLI, une librairie ou un package développé par quelqu'un plus malin que vous et moi. Voyons maintenant comment tirer profit de NPM.

Les différents modes d'installation des packages

NPM possède deux modes d'installation des packages : un mode local et un mode global. Concrètement, suivant le mode choisi, le répertoire d'installation sera différent et cela a une conséquence directe sur la manière dont Node charge ces packages par la suite.

En d'autres termes, un package installé en mode local sera accessible uniquement par votre application. Il sera local à votre projet. Alors qu'en mode global, ce package sera visible par tout votre système. Le mode global est donc plus adapté aux utilitaires et programmes CLI non destinés à être utilisés par une seule et unique application.

Le mode local

Le mode local est le mode par défaut lorsque vous installez un package. Dans ce mode, NPM installe tous les packages téléchargés dans le répertoire courant, celui de votre projet, et ne touche jamais aux préférences globales du système. Grâce à ce mode, vous pouvez ainsi contrôler application par application, quelle version du package utiliser. Vous pouvez par exemple avoir une application A qui utilise un package Foo en version X, et avoir une autre application B utilisant le même package Foo, mais en version Y.

Dans ce mode, NPM travaille dans le répertoire **./node_modules/** qu'il aura créé sous le répertoire courant. Par exemple, si vous êtes dans le

```
2. wchegham@wchegham-mbp-2: /tmp (zsh)
✘ wchegham 192.168.0.30  /tmp
$ nvm install 0.12
############################################################################ 100,0%
Now using node v0.12.4 (npm v2.10.1)
wchegham 192.168.0.30  /tmp
$ node -v
v0.12.4
wchegham 192.168.0.30  /tmp
$ npm -v
2.10.1
wchegham 192.168.0.30  /tmp
$
```

Fig.4

```
1. node --harmony (node)
wchegham 192.168.0.27  /tmp/programmez
$ node --harmony
> 1+1
2
> 'programmez!'.toUpperCase()
'PROGRAMMEZ!'
> process.env.PATH.split(':').filter( (str) => str.startsWith('/usr/local') )
[ '/usr/local/lib/node_modules/',
  '/usr/local/dart/dart-sdk/bin',
  '/usr/local/heroku/bin',
  '/usr/local/lib/node_modules/',
  '/usr/local/dart/dart-sdk/bin',
  '/usr/local/heroku/bin',
  '/usr/local/bin',
  '/usr/local/git/bin' ]
>
```

Fig.5

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ node --harmony test.js
2
PROGRAMMEZ!
[ '/usr/local/lib/node_modules/',
  '/usr/local/dart/dart-sdk/bin',
  '/usr/local/heroku/bin',
  '/usr/local/lib/node_modules/',
  '/usr/local/dart/dart-sdk/bin',
  '/usr/local/heroku/bin',
  '/usr/local/bin',
  '/usr/local/git/bin' ]
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.6

répertoire **/Users/wchegham/dev/app-A/**, NPM créera et utilisera le sous-répertoire **/Users/wchegham/dev/app-A/node_modules/** comme son répertoire de travail. Ainsi, lorsque vous lancerez votre application Node, ce dernier va d'abord chercher les dépendances dans **/Users/wchegham/dev/app-A/node_modules/** avant de remonter l'arborescence. Cela signifie que lorsque Node doit résoudre les dépendances, un package installé localement sera prioritaire par rapport au même package qui serait installé globalement.

Le mode global

Lorsqu'un package ou un CLI a été installé globalement, il sera stocké dans le répertoire suivant :

- Sous OS X et Linux (distribution classique) : **/usr/local/lib/node_modules/**
- Sous Windows 7 : **%AppData%\npm\node_modules**

Comme évoqué précédemment, ce mode est idéal pour installer des packages ou CLI qui sont destinés à être utilisés comme des programmes au niveau global de votre système. Pour installer un package en mode global, il suffit de l'activer via l'option **-g** lors de l'installation de celui-ci. Nous verrons des exemples plus parlants dans la suite de cet article. Si vous voulez en savoir plus sur ces deux modes d'installation, NPM propose la commande suivante qui donnera toutes les informations utiles :

```
$ npm help folders
```

Résoudre les éventuels problèmes de permissions :

Il peut arriver que vous ayez l'erreur suivante au moment d'installer un module globalement sur certains systèmes UNIX:

```
npm ERR! Error: EACCES, mkdir '/usr/local/lib/node_modules/
```

Cette erreur se produit fréquemment. Elle indique simplement que votre utilisateur n'a pas le droit d'écrire dans le répertoire **/usr/local/**. La solution la plus simple consiste à donner tous les droits à votre utilisateur sur les répertoires suivants :

```
$ sudo chown -R $USER /usr/local/lib/node_modules/
$ sudo chown -R $USER /usr/local/bin/
$ sudo chown -R $USER /usr/local/share/
```

Nous allons voir maintenant comment utiliser NPM pour installer un package.

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm install <nom-du-package>
```

Fig.7

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm install jshint -g
/usr/local/bin/jshint -> /usr/local/lib/node_modules/jshint/bin/jshint
jshint@2.8.0 /usr/local/lib/node_modules/jshint
├── strip-json-comments@1.0.2
├── exit@0.1.2
├── console-browserify@1.1.0 (date-now@0.1.4)
├── shelljs@0.3.0
├── minimatch@2.0.8 (brace-expansion@1.1.0)
├── cli@0.6.6 (glob@3.2.11)
├── lodash@3.7.0
└── htmlparser2@3.8.3 (domelementtype@1.3.0, entities@1.0.0, domhandler@2.3.0,
 readable-stream@1.1.13, domutils@1.5.1)
wchegham 192.168.0.27  /tmp/programmez
$ jshint -v
jshint v2.8.0
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.8

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm uninstall jshint -g
unbuild jshint@2.8.0
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.9

Installation et désinstallation des packages

Pour télécharger et installer la dernière version d'un package avec NPM, nous utilisons la commande suivante : Fig 7

Installons par exemple le package **jshint** globalement sur notre machine, grâce à la commande **npm install** : Fig 8

NPM a donc téléchargé la version 2.8.0 (la dernière version à cette date) puis l'a stocké dans le répertoire **/usr/local/lib/node_modules/** puis a positionné la CLI dans **/usr/local/bin/** . Suite à cela, nous pouvons utiliser la commande **jshint** simplement depuis partout dans notre terminal.

Pour désinstaller un package, rien de plus simple, nous utilisons la commande **npm uninstall** : Fig 9

Notez bien que j'ai inclus l'option **-g** dans la commande, cela est dû au fait que le package a été installé en mode global. Si le package a été installé en mode local, nous n'avons pas besoin d'inclure l'option **-g**.

Supposons maintenant que vous souhaitiez installer ce package localement (pour l'utiliser en tant que dépendance dans votre code par exemple) : Fig 10

Cela se fait de la même manière que précédemment, mais sans l'option **-g** bien évidemment. Mais dans ce cas, le package **jshint** a été installé dans le répertoire **node_modules/** de notre répertoire courant.

Avec NPM, il est également possible de préciser une version sémantique [3] en particulier d'un package à installer : Fig 11

De manière générale, la syntaxe est la suivante :

```
$ npm install <nom package>@<version sémantique>
```

Il existe plusieurs options permettant de préciser la version sémantique d'un package, je vous laisse découvrir à cette adresse — https://docs.npmjs.com/misc/semver#usage — toutes les combinaisons possibles et imaginables quant au choix de la version d'un package.

Mise à jour des packages installés

Il est bien évidemment possible de mettre à jour un package, grâce à la commande **npm update** : Fig 12. Notez que que si votre package est global, il faut ajouter l'option **-g**.

Utilisation du fichier package.json pour définir des dépendances

Lorsque nous développons une application Node, il est souvent coutume d'inclure, à la racine du répertoire projet, un fichier nommé package.json dans lequel nous définissons toutes les dépendances de notre application. Dans ce fichier, nous pouvons également renseigner d'autres informations

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm install jshint
jshint@2.8.0 node_modules/jshint
├── strip-json-comments@1.0.2
├── exit@0.1.2
├── console-browserify@1.1.0 (date-now@0.1.4)
├── shelljs@0.3.0
├── minimatch@2.0.8 (brace-expansion@1.1.0)
├── cli@0.6.6 (glob@3.2.11)
├── htmlparser2@3.8.3 (domelementtype@1.3.0, entities@1.0.0, domhandler@2.3.0,
 readable-stream@1.1.13, domutils@1.5.1)
└── lodash@3.7.0
wchegham 192.168.0.27  /tmp/programmez
$ tree -L 2
.
└── node_modules
    └── jshint

2 directories, 0 files
wchegham 192.168.0.27  /tmp/programmez
$ la
total 0
drwxr-xr-x   3 wchegham  wheel  102 28 jui 01:53 .
drwxrwxrwt  15 root      wheel  510 28 jui 01:32 ..
drwxr-xr-x   4 wchegham  wheel  136 28 jui 01:53 node_modules
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.10

comme le nom de l'application, la version, les auteurs...etc. Ces informations sont nécessaires si vous souhaitez publier votre application (ou package) sur NPM, mais ce n'est pas une obligation.
Le package.json est un fichier JSON — comme le suggère l'extension — qui contient une série d'attributs et ressemble à ceci : Fig 13
Pour créer ce fichier, il suffit de lancer la commande **npm init** dans votre répertoire projet : Fig 14
Le programme vous posera une série de questions puis créera le fichier.
Pour indiquer à NPM la liste des dépendances de votre application, nous pouvons demander à NPM de l'ajouter automatiquement pour nous au moment d'installer cette dépendance, soit nous l'ajoutons à la main.
Pour que NPM puisse ajouter une dépendance automatiquement, il suffit d'indiquer l'option **–save** au moment de l'installation : Fig 15
Lorsque nous utilisons le fichier package.json pour déclarer les dépendances de notre application, il suffit d'exécuter la commande **npm install (sans paramètres)** à la racine du répertoire où se trouve le fichier package.json, NPM va alors lire automatiquement ce fichier puis installera toutes les dépendances qui y sont indiquées.
Ainsi, pas besoin de mettre tout le répertoire **node_modules** sur votre système de contrôle des versions (GIT, SVN...). Cela marche également avec la mise à jour : **npm update (sans paramètres)**.
Voilà ! vous avez toutes les billes pour commencer à utiliser NPM dans votre projet Node. Sachez également que NPM propose encore beaucoup de fonctionnalités que nous n'avons pas évoquées dans cette phase de découverte.
Maintenant que nous avons installé Node et NPM, nous voilà prêts pour découvrir les incarnes de Node à travers la présentation de quelques concepts propres à Node.

LES INCARNES DE NODE

Comme vous devez sans doute le savoir, lorsque nous parlons d'une application "modulaire", nous parlons d'une application composée de plusieurs blocs de fonctionnalités distinctes appelées "modules". Ce patron de découplage [4] facilite la maintenabilité de nos applications en enlevant les dépendances là où c'est possible.
Cependant, à la différence des autres langages de programmation traditionnels, la version actuelle de JavaScript (ES5) ne propose pas de mécanismes de création ou d'import de modules. Heureusement, Node implémente le standard CommonJS, qui apporte ce genre de mécanisme à JavaScript.
À noter qu'il existe un autre standard de définition et de chargement de modules pour JavaScript : AMD (Asynchronous Module Definition). Cet

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm install jshint@2.5.x
jshint@2.5.11 node_modules/jshint
├── strip-json-comments@1.0.2
├── underscore@1.6.0
├── console-browserify@1.1.0 (date-now@0.1.4)
├── exit@0.1.2
├── minimatch@1.0.0 (sigmund@1.0.1, lru-cache@2.6.4)
├── shelljs@0.3.0
├── cli@0.6.6 (glob@3.2.11)
└── htmlparser2@3.8.3 (domelementtype@1.3.0, entities@1.0.0, domhandler@2.3.0, readable-stream@1.1.13, domutils@1.5.1)
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.11

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm update jshint
jshint@2.8.0 node_modules/jshint
├── strip-json-comments@1.0.2
├── exit@0.1.2
├── console-browserify@1.1.0 (date-now@0.1.4)
├── shelljs@0.3.0
├── minimatch@2.0.8 (brace-expansion@1.1.0)
├── cli@0.6.6 (glob@3.2.11)
├── lodash@3.7.0
└── htmlparser2@3.8.3 (domelementtype@1.3.0, entities@1.0.0, domhandler@2.3.0, domutils@1.5.1, readable-stream@1.1.13)
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.12

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ catt package.json
{
  "name": "programmez",
  "version": "1.0.0",
  "description": "Node.js de A à  Z",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Wassim Chegham <wassim.chegham@gmail.com>",
  "license": "ISC"
}
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.13

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (programmez) programmez
version: (1.0.0) 1.0.0
description: Node.js de A à  Z
entry point: (index.js)
test command:
git repository:
keywords:
author: Wassim Chegham <wassim.chegham@gmail.com>
license: (ISC)
About to write to /private/tmp/programmez/package.json:

{
  "name": "programmez",
  "version": "1.0.0",
  "description": "Node.js de A à  Z",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Wassim Chegham <wassim.chegham@gmail.com>",
  "license": "ISC"
}


Is this ok? (yes) yes
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.14

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ npm install jshint@2.5.x --save
npm WARN package.json programmez@1.0.0 No repository field.
npm WARN package.json programmez@1.0.0 No README data
jshint@2.5.11 node_modules/jshint
├── strip-json-comments@1.0.2
├── underscore@1.6.0
├── exit@0.1.2
├── console-browserify@1.1.0 (date-now@0.1.4)
├── shelljs@0.3.0
├── minimatch@1.0.0 (sigmund@1.0.1, lru-cache@2.6.4)
├── cli@0.6.6 (glob@3.2.11)
└── htmlparser2@3.8.3 (domelementtype@1.3.0, entities@1.0.0, domhandler@2.3.0, readable-stream@1.1.13, domutils@1.5.1)
wchegham 192.168.0.27  /tmp/programmez
$ catt package.json
{
  "name": "programmez",
  "version": "1.0.0",
  "description": "Node.js de A à  Z",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Wassim Chegham <wassim.chegham@gmail.com>",
  "license": "ISC",
  "dependencies": {
    "jshint": "^2.5.11"
  }
}
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.15

autre standard a été propulsé par Dojo Toolkit [5] ou encore Require.js [6]. Il est vrai qu'avec le temps, les deux standards se sont un peu spécialisés : AMD côté client, et CommonJS côté serveur. Mais vous être libres d'utiliser l'un ou l'autre. Veuillez noter tout de même que la toute nouvelle version de JavaScript (ES6 — ES2015) apporte enfin un système [7] de définition et de chargement de modules qui manquait tant au langage. Ce nouveau format est censé devenir LE standard avec la démocratisation de ES2015. En attendant, pour nos applications Node, nous utiliserons CommonJS pour définir et charger nos modules.

La gestion des modules dans Node

Dans Node, les modules sont référencés soit par leurs noms soit par leurs chemins. Un module référencé par son nom correspond éventuellement à un fichier, sauf pour les modules "built-in". Node utilise ces modules "built-in" pour exposer les fonctionnalités systèmes au développeur. D'autres modules incluent également des packages récupérés depuis Internet par NPM ou localement (modules développés par vous ou vos collègues).

Chargement d'un module

Tout module expose une API que le développeur peut appeler pour résoudre un problème. Pour utiliser un module, nous devons le charger via la fonction **require()** : Fig 16

Ceci va importer les packages :

- **HTTP** qui est un package "built-in" ;
- **jshint** le package installé par NPM ;
- **Programmez**, un exemple de module que nous avons "développé". Notez que pour charger un module appartenant au projet, nous devons préciser le chemin vers ce fichier !

La fonction **require()** retourne un objet représentant l'API JavaScript exposée par le module en question. En fonction du module, cet objet peut être n'importe quel type JavaScript : une fonction, un tableau, une chaîne de caractères, etc.

Création d'un module

Avec Node, la création de modules est chose aisée, ce qui je pense explique d'ailleurs le succès de NPM [8]. Pour créer un module, il suffit de créer un fichier JavaScript et d'utiliser l'instruction **module.exports** pour exposer une API. Par exemple : Fig 17

Fig.16

```
// app.js
var http = require('');
var jshint = require('jshint');
var programmez = require('./lib/programmez');
```

Line 4, Column 46; Saved /private/tmp/programmez/app.js (UTF-8) — Tab Size: 2 — JavaScript

Fig.17

```
// app.js
var programmez = require('./lib/programmez');

console.log(programmez.title); // Programmez!
console.log(programmez.getCurrentIssue()); // 187
console.log(programmez.issue); // undefined
```

```
// programmez.js
var issue = 187;

module.exports = {
  title: 'Programmez!',
  getCurrentIssue: () => issue
}
```

Line 5, Column 36 — Tab Size: 2 — JavaScript

Dans cet exemple, nous avons créé un module **programmez** dans le fichier **programmez.js**. Dans ce module, nous avons choisi d'exposer au monde un objet JavaScript contenant deux attributs : **title** de type chaîne de caractères, et une fonction **getCurrentIssue()**.

Dans notre fichier **app.js**, nous avons importé le module **programmez**, puis nous avons accès à ces deux valeurs.

Notez que la variable issue dans le fichier **programmez.js** n'a pas été exportée, elle n'est donc pas accessible en dehors du module !

Notez également que nous aurions pu exporter l'API en plusieurs fois, comme ceci : Fig 18

C'est une syntaxe valide. Personnellement, je préfère exporter l'API d'un module en une seule fois, cela rend les choses plus claires lorsqu'on développe des applications complexes.

Mise en cache des modules

Un point très important à connaître lorsque nous travaillons avec les modules dans Node : lorsque nous appelons la fonction **require()** pour importer un module, Node charge le module, exécute le code de ce dernier puis le met en cache [9]. Ce qui signifie que tous les appels suivants de **require()** vers ce "même" module, retourneront la version mise en cache. Autrement dit, le code d'un module est interprété une seule fois !

Voilà ! c'est aussi simple que cela. Nous verrons dans la suite de cet article, d'autres exemples d'utilisations et définitions de modules.

Manipulation des données binaires grâce aux Buffers

Nous allons tenter de comprendre comment manipuler des données binaires avec Node.

En effet, JavaScript a été initialement créé pour manipuler des chaînes de caractères, dans des documents HTML et n'a donc pas été conçu pour manipuler des données binaires.

Node quant à lui, doit gérer à la fois des sources de données textuelles telles que HTTP, mais aussi des sources de données binaires telles que des bases de données, les manipulations des images et la gestion des uploads de fichiers binaires. Pour faciliter ces traitements, Node inclut de base une pseudo-class **Buffer** exposant une API JavaScript.

Pour rappel, un buffer est une zone de la mémoire physique qui est utilisée pour stocker des données temporaires. Dans Node, chaque buffer correspond à une certaine quantité de mémoire allouée en dehors de V8, ce qui est intéressant puisque le ramasse-miettes ne pourra pas agir dessus.

Aussi, un buffer ne possède pas d'encodage, ce qui signifie que sa taille est fixe et connue. Les chaînes de caractères, quant à elles, sont concernées par les encodages, comme UTF-8 — qui stocke en interne plusieurs caractères étrangers sous forme d'une séquence d'octets. La manipulation des chaînes de caractères prend systématiquement un encodage en compte et suivant l'encodage utilisé, un caractère peut être représenté grâce à un ou plusieurs octets. Ceci pose donc problème pour les données binaires, puisqu'elles ne sont pas encodées sous forme de caractères, mais plutôt d'octets — mais elles peuvent contenir des sous-séquences pouvant accidentellement être interprétées comme un caractère UTF-8. Voyons maintenant comment créer un buffer.

Fig.18

```
// programmez.js
var issue = 187;

module.exports.title = 'Programmez!';
module.exports.getCurrentIssue = () => issue;
```

Line 4, Column 46 — Tab Size: 2 — JavaScript

Création d'un buffer

Pour créer un buffer, il suffit d'instancier la pseudo-class **Buffer** : Fig 19

Il est possible de créer un buffer avec un contenu initial, ou bien juste en précisant une taille en octets à allouer, voire un tableau d'octets dans certains cas. Il est également possible de spécifier un encodage en particulier : UTF-8 (encodage par défaut), UTF16LE, BASE64, HEX ou ASCII.

Notez que dans le cas où nous créons un buffer avec une taille donnée, et sans contenu initial, les données du buffer ne sont pas initialisées et seront donc aléatoires lors de sa création (les données présentent à cet instant-là dans la zone mémoire tampon) !

La pseudo-class **Buffer** propose la méthode **toString()** qui permet de décoder un buffer en chaîne de caractère JavaScript, en précisant ou non un encodage particulier parmi ceux cités précédemment.

Manipulation d'un buffer

Après avoir créé un buffer, il serait intéressant d'interagir avec, en manipulant son contenu. Tentons d'écrire dans un buffer contenant des caractères UTF-8 : Fig 20

Nous pouvons utiliser la méthode **write()** pour écrire directement dans un buffer, nous précisant donc la chaîne à placer dans le buffer, et un paramètre optionnel qui est le nombre d'octets correspondant à l'emplacement du début d'écriture dans le buffer.

Node propose également la possibilité de copier des buffers ou simplement des bouts dans d'autres buffers : Fig 21

Dans cet exemple, nous avons créé deux buffers de 11 octets chacun. Nous avons écrit dans le premier. Ensuite, nous avons copié le contenu de buffer1 — en commençant par le 3e octet — dans buffer2 — à partir du 3e octet. Ceci explique le fait qu'à ce stade, les 3 premiers octets de buffer2 contiennent des données aléatoires se trouvant dans la zone mémoire allouée par V8. Lors de la dernière étape, nous écrivons dans buffer2 l'équivalent de 3 octets de données à partir du début, ce qui vient donc compléter la zone allouée pour buffer2. Au final, nous avons bien la chaîne décodée : **Pro**grammez! sans la couleur rouge bien sûr ;)

Grâce à la gestion des buffers offerte par Node, nous avons la possibilité de manipuler directement le contenu de la mémoire depuis JavaScript : simple et efficace !

Fig.19

```
programmez.js

app.js
var programmez = require('./lib/programmez');
programmez.run();

programmez.js
module.exports.run = () => {

  var buffer1 = new Buffer('Programmez!', 'utf-8');
  var buffer2 = new Buffer('UHJvZ3JhbW1leiE=', 'base64');
  var buffer3 = new Buffer(11); // 11 bytes

  console.log(buffer1.toString('base64')); // UHJvZ3JhbW1leiE
  console.log(buffer2.toString('utf-8')); // Programmez!
  console.log(buffer3.toString()); // ���_�(}

};

Line 1, Column 29                                  Tab Size: 2   JavaScript
```

Fig.20

```
programmez.js

app.js
var programmez = require('./lib/programmez');
programmez.run();

programmez.js
module.exports.run = () => {

  var buffer1 = new Buffer(22); // 22 octets
  buffer1.write('Programmez!');
  buffer1.write('Programmez!', 11); // 11 octets offset

  console.log(buffer1.toString()); // Programmez!Programmez!

};

2 selection regions                                Tab Size: 2   JavaScript
```

La gestion des évènements et les timers

Dans le monde de Node, la plupart des objets émettent des évènements. Par exemple, lors de la lecture d'un fichier, le flux de lecture émet un évènement "data" à chaque fois qu'un bout du fichier a été lu. De la même manière, un serveur TCP émet un événement "connect" à chaque fois qu'un client se connecte. Nous appelons ces objets dans la nomenclature Node, des "event emitters". Ces "event emitter" permettent au développeur d'attacher des actions à des événements, bien définis ou personnalisés.

Nous pouvons "presque" faire l'analogie avec les évènements produits dans un navigateur — ou une UI de manière générale. Le navigateur émet différents "event" suite à différents évènements, par exemple : la fin de chargement de la page, lorsque l'utilisateur interagit avec le clavier ou la souris, etc.

J'ai dit "presque", car le modèle des "event emitter" de Node est basé sur le pattern publish/subscribe. Ce pattern est donc implémenté nativement par Node, or côté navigateur, nous retrouvons des implémentations au sein de certains frameworks et librairies, tels que PubSubJS [10]. Ce pattern est simple à implémenter, voyez par vous-même à travers ce code que j'ai écrit [11].

De plus, avec Node, il est également possible de créer ses propres évènements — comme dans le navigateur d'ailleurs.

L'intérêt d'utiliser les "event emitters" dans nos applications Node, permet tout simplement d'éviter de créer un couplage fort entre les différents composants de notre application, et donc d'éviter l'anti-pattern plat de spaghettis [12].

Ces fonctionnalités des "event emitters" sont proposées par la pseudo-class **EventEmitter,** qui est fournie par le module **event**.

Utilisation de EventEmitter

Exemple basique

Pour utiliser la classe **EventEmitter**, rien de plus simple. Voici un exemple très basique. Commençons par charger le module **events** : Fig 22

Ensuite, nous créons une instance de **EventEmitter**, grâce à laquelle nous allons pouvoir enregistrer des callbacks sur des évènements. Par exemple, nous avons enregistré sur un événement "pro", un callback représenté par la fonction console.log, qui affichera dans la sortie standard le texte que nous allons émettre. Pour cela, nous avons utilisé la méthode **on()**.

Maintenant, pour déclencher l'exécution de la callback, il suffit d'émettre l'évènement utilisé lors de l'enregistrement. Pour cela, nous utilisons la méthode **emit()**.

Fig.21

```
programmez.js

app.js
var programmez = require('./lib/programmez');
programmez.run();

programmez.js
module.exports.run = () => {

  var buffer1 = new Buffer(11);
  var buffer2 = new Buffer(11);
  buffer1.write('Programmez!');
  console.log(buffer1.toString()); // Programmez!

  buffer1.copy(buffer2, 3, 3);
  console.log(buffer2.toString()); // ���grammez!

  buffer2.write('PRO');
  console.log(buffer2.toString()); // PROgrammez!

};

Line 12, Column 50                                 Tab Size: 2   JavaScript
```

Note :
Bien évidemment, il est possible d'enregistrer plusieurs callback sur un évènement. Tous ces callback seront invoqués lorsque l'évènement sera déclenché. Par défaut, Node limite ce nombre à 10, mais il est possible de modifier ce max via la méthode **setMaxListeners(n).**

Exemple avancé

Nous allons maintenant voir un autre exemple, encore plus intéressant : Fig 23

Dans le fichier **programmez.js**, nous implémentons un prototype de classe **Programmez** qui hérite de celui du module **event** (est donc **EventEmitter**). Ainsi, **Programmez** hérite de toutes les méthodes de **EventEmitter** : on(), emit() ,etc.

Programmez déclenche un événement "pro" toutes les une seconde. Évidemment, cette abstraction ne fait rien d'exceptionnel, mais nous pouvons imaginer qu'elle est responsable de lire le contenu d'un fichier, interroger une ressource REST ou simplement accéder à une base de données. Mais le plus intéressant, c'est qu'elle hérite du comportement de **EventEmitter**. Ce qui implique que de la même manière qu'un serveur TCP qui émettrait l'événement "connect" lorsqu'un nouveau client se connecte, notre pseudo-class émet quant à elle, un événement personnalisé nommé "pro". Du côté de notre **programmez** principal **app.js**, nous avons donc la possibilité d'enregistrer un callback sur l'événement "pro", ce que nous faisons à la ligne 4. Ceci produit le résultat suivant : Fig 24

Voilà ! Vous avez toutes les billes pour commencer à utiliser les "event emitter" dans vos applications Node. Ce pattern est très utilisé dans Node, je vous conseille vivement de vous familiariser avec, et de l'adopter sans hésiter. Nous allons maintenant passer à l'étape suivante qui traitera des processus, des "Streams" (ou flux de données), des accès réseau, et à la gestion des fichiers dans Node.

STREAMS, PROCESSUS, FICHIERS.

Lorsque nous développons une application Node, celle-ci sollicite tout un tas d'API offertes par Node, que ce soit lire ou écrire dans un fichier, accéder au réseau, interagir avec des processus systems, etc. Découvrons ensemble ces API.

Les Streams

Node possède deux types d'abstractions des flux de données : les flux d'écriture et les flux de lecture, représentant les données entrantes et sortantes. Ces abstractions sont implémentées à travers différents objets dans Node (comme pour les "event emitter"). Un exemple de flux de données est un socket TCP vers lequel nous pouvons envoyer des données (écriture) ou récupérer des données (lecture). Le module "fs" est un autre exemple d'un objet implémentant l'API des Streams.

Les flux de lecture

Un flux de lecture est comme un robinet, nous pouvons contrôler le débit en ouvrant ou fermant ce robinet.

Avec Node, nous pouvons être notifié lorsque des données sont disponibles ou bien lorsqu'il n'y plus de données à lire. Pour rappel, les Streams sont des "event emitter" !

Les Streams envoient les données par petits paquets. En écoutant l'évènement "data", nous pouvons être notifiés de l'arrivée de chaque paquet. Par défaut, les données que nous lisons depuis un flux de lecture sont des buffers. Il est de notre responsabilité d'indiquer à Node que nous souhaitons avoir ces données sous forme de chaîne de caractères, en utilisant la méthode **setEncoding().** Voici un exemple illustrant mes propos : Fig 25

Nous récupérons un flux de lecture depuis une source de données (un fichier, une socket TCP...etc.). Nous écoutons l'évènement "data" pour lire les paquets de données qui arrivent, sous format binaire pour **readableStream1**.

Pour **readableStream2**, les données reçues sont sous format textuel, car nous avons demandé à Node de les encoder en UTF8.

Il est également possible d'exécuter des actions lorsqu'il n'y a plus de paquets, en écoutant l'événement "end" : Fig 26

Après la réception de l'évènement "end", nous ne recevrons plus l'évènement "data". Ce qui est normal puisqu'il n'y a plus de données à lire.

Fig.22

```
// app.js — programmez
var programmez = require('./lib/programmez');
programmez.run();
```

```
// programmez.js
var EventEmitter = require('events');

module.exports.run = () => {

  var evt = new EventEmitter();
  evt.on('pro', console.log);
  evt.on('pro', (str) => console.log(str.toUpperCase()));
  evt.emit('pro', 'programmez!'); // programmez! & PROGRAMMEZ!

};
```

Line 2, Column 18 — Tab Size: 2 — JavaScript

Fig.23

```
// app.js — programmez
1 import {Programmez} from './lib/programmez';
2 
3 var programmez = new Programmez();
4 programmez.on('pro', (str) => console.log('programmez!') );
```

```
// programmez.js
1 export class Programmez extends require('events') {
2   constructor(){
3     super()
4     setInterval(()=> this.emit('pro'), 1000);
5   }
6 }
```

Line 1, Column 45; Saved /private/tmp/programmez/app.js (UTF-8) — Tab Size: 2 — JavaScript

Fig.24

```
2. wchegham@wchegham-mbp-2: /tmp/programmez...
wchegham 192.168.0.27  /tmp/programmez
$ ./b.sh
programmez!
programmez!
programmez!
programmez!
programmez!
^C
✘ wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.25

```
// untitled
var readableStream1 = /* Stream */
readableStream1.on('data', (data) => {
  // data is a buffer;
  console.log('data:', data);
});

var readableStream2 = /* Stream */
readableStream2.setEncoding('utf8');
readableStream2.on('data', (data) => {
  // data is a utf8-encoded string;
  console.log('data:', data);
});
```

Line 1, Column 1 — Tab Size: 2 — JavaScript

Les flux d'écriture

Parallèlement aux flux de lecture, il y a les flux d'écriture. Un flux d'écriture est un flux vers lequel nous pouvons envoyer des données, ou donc, écrire. Cela peut être un fichier, une connexion TCP.

Écrire dans un flux d'écriture est très simple : Fig 27

Il est possible d'écrire du texte. Par défaut, c'est de l'UTF-8, mais nous pouvons préciser un second paramètre pour indiquer un encodage différent, BASE64 par exemple.

Il est également possible d'écrire un buffer.

À chaque écriture dans le flux, Node peut, soit passer immédiatement les données vers la “mémoire kernel”, soit (pour une raison X ou Y), il peut stocker ces données dans une file d'attente, dans la zone mémoire allouée au processus courant. Lorsque Node traite les données stockées dans cette file, il émet un évènement “drain” que nous pouvons écouter : Fig 28

Étudions ce phénomène de plus près.

Le problème des “clients lents”

À chaque fois qu'un processus a besoin de lire de données, puis doit les envoyer vers un consommateur, il peut se produire ce que l'on appelle le problème du “client lent”.

Node ne bloque pas pour les opérations d'E/S. Cela signifie qu'il ne bloque pas lorsqu'il lit ou écrit. Il met en buffer si la méthode **write()** n'arrive pas écrire les données dans la “mémoire kernel”.

Imaginons le scénario suivant, vous êtes en train de lire des données depuis un flux de lecture (un fichier par exemple), et vous les envoyez vers un flux d'écriture (une liaison TCP vers un navigateur par exemple). Le fichier que vous lisez est en local, la lecture est donc rapide, mais la connexion réseau quant à elle est relativement lente — même si c'est de la fibre ^^. Ce qui va se passer est que vous allez lire plus que ce que vous allez envoyer. Node va donc commencer à mettre ce superflu de données en buffer. Pire, Node mettra en buffer les données de ce fichier pour chaque appel à la fonction **write()**. Multipliez cela par le nombre de requêtes et vous voilà avec un énorme souci de mémoire ! Heureusement, Node nous offre un moyen pour contrôler l'arrivée du flux de données.

Prenons cet exemple : Fig 29

Dans cet exemple, nous avons lancé un serveur HTTP qui envoie en “streaming” le contenu du fichier programmez.txt vers un client Web. Ce qui est important dans cet exemple : lors de la réception de l'évènement “data”, nous vérifions que le contenu a bien été écrit par la méthode **write()**, dans le cas contraire, nous mettons en pause la lecture du fichier. Puis, lorsque Node finit par envoyer le contenu qu'il a mis en buffer — suite à l'évènement “drain” donc — nous remettons la lecture en route.

Ce phénomène de “pause/resume” des flux est un pattern très répandu dans Node. Node l'a donc implémenté dans la méthode **pipe()**.

Voici l'exemple précédent réécrit en utilisant la méthode **pipe()** : Fig 30

Processus

Node a été conçu pour les E/S de manière efficace, mais il peut arriver dans certains cas que des applications Node exécutent des tâches intensives en calcul CPU. Généralement ces tâches finissent par bloquer la boucle d'évènements, ce qui a pour conséquence de bloquer l'exécution de votre application, tout simplement. De la même manière que lorsque vous effectuez des traitements très lourds dans un navigateur, votre page arrête de répondre. Dans ce genre de cas, il vaut mieux privilégier les Web Workers pour ce genre de tâches, mais ce sujet mériterait son propre article. Avec Node, nous avons la possibilité de démarrer des processus fils depuis notre application. Un processus fils possède un canal de communication bidirectionnelle avec son processus parent. Ces processus fils peuvent aussi servir à lancer des programmes externes, ou exécuter des commandes système, donc en dehors de notre application.

Exécution de commandes externes

Pour pouvoir lancer des commandes externes, nous devons importer le module **child_process**.

Ce dernier nous fournit plusieurs méthodes, dont la méthode **exec()**, celle qui permet d'exécuter des programmes externes : Fig 31

Cette méthode prend en paramètres, la commande ou le nom du pro-

Fig.26

```
var readableStream = /* Stream */
readableStream
  .on('data', (data) => {
    console.log('data:', data);
  })
  .on('end', () => {
    console.log('no more data');
  });
```

Fig.27

```
var writableStream = /* Stream */

writableStream.write('Programmez!');

writableStream.write('UHJvZ3JhbW1leiE=', 'base64');

writableStream.write(new Buffer('Programmez!'));
```

Fig.28

```
var writableStream = /* Stream */

writableStream.on('drain', () => {
  console.log('drain event');
});
```

Fig.29

```
ProgrammezServer( (request, response) => {

  var rs = require('fs').createReadStream('programmez.txt');
  rs.on('data', (data) => {
    if(!response.write(data)){
      rs.pause();
    }
  }).on('end', () => response.end() );

  response.on('drain', () => rs.resume() );

});
```

Fig.30

```
ProgrammezServer( (request, response) => {

  require('fs')
    .createReadStream('programmez.txt')
    .pipe(response);

});
```

gramme à exécuter, un ensemble d'options, et une fonction de callback. Ce callback nous renvoie trois paramètres : une **erreur** (éventuellement), le contenu de la sortie standard **stdout**, le contenu de la sortie d'erreur **strerr**. Fig 32

Nous pouvons aussi passer des options à la méthode **exec()** afin de paramétrer la commande exécutée. Nous pouvons, par exemple, définir le répertoire de travail de la commande, le type du shell cible, positionner des variables d'environnement, etc. Je vous laisse consulter la documentation pour plus de détails.

Si la commande retourne une erreur, le premier paramètre **erreur** serait une instance de la classe **Error**. Nous pouvons alors accéder au code d'erreur, le signal d'interruption, et la trace d'erreur. Les contenus **stdout** et **stderr** sont quant à eux des buffers. Voici un exemple d'un cas où l'exécution d'une commande **fake**, par exemple, retourne une erreur : Fig 33

Contrôler le cycle de vie d'un processus

Avec la méthode **exec()**, il n'est pas possible d'avoir un tel contrôle. Il n'y a aucune communication possible entre le processus parent et le processus fils. De plus, le résultat **stdout** retourné par **exec()** n'est pas un flux, il est mis en buffer ce qui impose certaines limitations. Il est souvent utile de pouvoir contrôler le cycle de vie du processus que nous lançons. Pour cela, Node nous propose la méthode **spawn()** : Fig 34

La méthode **spawn()** exécute la commande passée en paramètre et retourne un objet **ChildProcess**. À partir de cet objet, nous avons accès aux sorties standard **stdout** et sortie d'erreur **stderr** du processus fils. Ces deux sorties sont des flux de lecture, sur lesquels nous pouvons écouter l'évènement **data**, qui nous permet de lire les paquets qui sont retournés au fur et à mesure par le processus. Grâce à la méthode **spawn()**, il nous est possible de communiquer avec le processus fils, via l'entrée standard **stdin**, qui plus est, est un flux d'écriture. Voyons un exemple : Fig 35

Dans cet exemple, nous exécutons un programme Node, child.js, dans un processus fils, ensuite nous lui envoyons des données via l'entrée standard **stdin**.

Pour arrêter un processus enfant, nous lui envoyons un signal d'interruption d'arrêt brutal **SIGKILL**. Ici, nous envoyons le signal après une seconde. Bien évidemment vous avez la possibilité d'envoyer n'importe quel autre signal qui sera plus adapté à votre situation. En effet, il existe une multitude de signaux, représentés chacun par un code unique, et chaque code a une signification. La plupart de ces signaux ont pour but de tuer les processus. Lorsqu'un processus reçoit un signal qu'il ne peut pas traiter, il est immédiatement terminé. Certains signaux sont destinés à être traités par les processus fils, alors que d'autres peuvent être traités par le système d'exploitation uniquement.

Pour envoyer un signal, nous utilisons la méthode **kill()**, en lui précisant le code du signal. Par défaut, cette méthode, lorsqu'elle est appelée sans code, envoie un code **SIGTERM**. Attention tout de même, ce n'est pas parce que cette méthode s'appelle **kill()** qu'elle permet de tuer un processus fils. Un processus fils a la possibilité de surcharger le code du signal envoyé. Cette technique permet en effet à un processus fils de faire le ménage avant qu'il ne soit tué; par exemple, fermer la connexion d'une base de données. Cette règle ne s'applique pas aux signaux **SIGKILL** et **SIGSTOP**, qui sont la chasse gardée de l'OS.

Voilà ! vous pouvez maintenant lancer et interagir avec des processus fils qui vous permettront d'éviter de bloquer le processus principal et donc de bloquer l'exécution de votre application.

Passons maintenant à une autre tâche récurrente dans une application Node qui est l'accès et la manipulation des fichiers.

Fig.31

```
programmez.js — programmez

app.js
'use strict';

var programmez = require('./lib/programmez');
programmez.run();

programmez.js
'use strict';

var child = require('child_process');
var options = {/* cwd, env, shell, maxBuffer ... */};
module.exports.run = () => {
  child.exec('date', options, (error, stdout, stderr) => {

    if (error) {
      console.log(error.stack);
      console.log('Code    : ', error.code);
      console.log('Signal  : ', error.signal);
    }
    console.log('Stdout  :\n', stdout);
    console.log('Stderr  :\n', stderr);
  }).on('error', (code) => {
    console.log('Child exit: ', code);
  });
};

Line 1, Column 1    Tab Size: 2    JavaScript
```

Fig.32

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ ./run.sh
Stdout  :
 Dim 12 jul 2015 05:08:37 CEST

Stderror:

wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.33

```
1. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27  /tmp/programmez
$ ./run.sh
Error: Command failed: /bin/sh -c fake
/bin/sh: fake: command not found

    at ChildProcess.exithandler (child_process.js:751:12)
    at ChildProcess.emit (events.js:110:17)
    at maybeClose (child_process.js:1015:16)
    at Socket.<anonymous> (child_process.js:1183:11)
    at Socket.emit (events.js:107:17)
    at Pipe.close (net.js:485:12)
Code    :  127
Signal  :  null
Stdout  :

Stderr  :
 /bin/sh: fake: command not found

wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.34

```
programmez.js — programmez

app.js
'use strict';

var programmez = require('./lib/programmez');
programmez.run();

programmez.js
'use strict';

var child = require('child_process');
var options = {/* cwd, env, shell, maxBuffer ... */};
module.exports.run = () => {
  var command = child.spawn('date');
  command.stdout.on('data', (output) => console.log('Stdout: ', output.toString()));
  command.stderr.on('data', (error) => console.log('Stderr: ', error.toString()));
};

Line 5, Column 28    Tab Size: 2    JavaScript
```

Fig.35

```
programmez.js — programmez

app.js
'use strict';

var programmez = require('./lib/programmez');
programmez.run();

programmez.js
'use strict';

var child = require('child_process');
module.exports.run = () => {
  var command = child.spawn('node', ['lib/child.js']);
  command.stdin.write('Programmez!');
  command.stdout.on('data', (output) => console.log('Child replies: ', output.toString()));
  command.stderr.on('data', (error) => console.log('Stderr: ', error.toString()));
  command.on('exit', (code) => console.log('Child exited with code: ', code));
  setTimeout(() => command.kill('SIGKILL'), 1000);
};

child.js
'use strict';

process.stdin.resume();
process.stdin.on('data', (data) => {
  console.log('Getting data from parent process: ', data.toString());
});

Line 1, Column 14    Tab Size: 2    JavaScript
```

Système de fichiers

Node offre une API très pratique de manipulation des fichiers, comme s'ils étaient des flux, Streams. Mais cette API ne vous permet pas de vous déplacer à l'intérieur d'un fichier, aller directement à une position précise par exemple. Pour cela, il faut descendre un peu plus bas dans les couches, au niveau du système de fichiers (**File System**).

Un système de fichier définit comment les fichiers doivent être organisés, comment ils sont identifiables, l'endroit où ils sont stockés, leurs propriétés...etc. Les systèmes de fichiers classiques sont UFS (UNIX File System) pour UNIX, NTFS (New Technology File System) pour Windows et HFS+ (Hierarchical File System Plus) pour OS X.

L'API de Node quant à elle est inspirée de celle d'UNIX — et repose sur les spécifications POSIX (Portable Operating System Interface for UNIX).

Manipulation des chemins

Comme vous le savez bien, les chemins des fichiers peuvent être soit relatifs soit absolus. Il est possible de les concaténer, en extraire les noms des fichiers ou encore vérifier l'existence d'un fichier. Ces chemins de fichiers sont représentés par des chaînes de caractère, et les manipuler tels quels peut rapidement conduire à des erreurs. Par exemple, les séparateurs de fichiers ne sont pas les mêmes d'un OS à l'autre. Heureusement pour nous, Node possède un module **path** qui nous fournit une API pour gérer ces différents cas tordus. Donc, je vous recommande vivement de déléguer toutes les manipulations des chemins à ce module.

Tout de même, sachez que le module **path** se contente de manipuler les chemins, il n'interagit pas avec le système de fichiers.

Voici un exemple de code rassemblant la plupart des méthodes fournies par ce module : Fig 36

Explications de ces méthodes :

- Ligne 7 : **normalize(C)** permet d'avoir un chemin C "normalisé" et prend soin de nettoyer les .. et / superflus,
- Ligne 8 : **join()** permet de concaténer tous les chemins fournit et retourne un chemin normalize,
- Ligne 9 : **relative(A, B)** permet de retourner le chemin relatif pour aller de A vers B,
- Ligne 10 : **resolve(A, B, ..., Y, Z)** permet de retourner le chemin absolu correspondant au chemin Z. Les autres chemins A à Y, sont les chemins utilisés lors de la normalisation (construction) du chemin absolu resultant,
- Ligne 11 : **extname(F)** permet de retourner l'extension du fichier F,
- Ligne 12 : **basename(F)** permet d'avoir le dernier segment du chemin du fichier F,
- Ligne 13 : **dirname(F)** permet d'extraire le nom du répertoire du fichier F,

Manipulation des fichiers

En plus de manipuler les chemins des fichiers, Node nous offre le module **fs**, pour manipuler les fichiers. Nous pouvons accéder aux attributs des fichiers, les ouvrir, les lire, écrire dedans et les fermer.

Accès aux attributs

Si vous avez besoin de manipuler des fichiers, par exemple pour une application d'upload de documents, le module **fs** propose la méthode **stat()** que vous pouvez utiliser comme ceci : Fig 37

Ce qui produit le résultat suivant : Fig 38

Ce qui est important à savoir est que l'objet **stats** retournées est une instance de la classe **fs.Stats.** Cette classe offre des méthodes intéressantes telles que **isFile()**, **isDirectory()** ou encore **isSymbolicLink()**...etc.

Ouvrir, lire, écrire et fermer un fichier

Avant de pouvoir manipuler un fichier, il est nécessaire de l'ouvrir avec la méthode **fs.open()**. Cette méthode accepte un callback qui sera invoqué avec le descripteur de fichier, grâce auquel nous allons lire le fichier ou écrire dedans.

Une fois le fichier ouvert, nous pouvons lire une partie, mais pour cela il nous faut créer un Buffer qui va contenir les données. Ce Buffer va être transmis à la méthode **fs.read()** pour accueillir les données. Voyons cet exemple : Fig 39

Ici nous avons ouvert le fichier **app.js** (ligne 6) avec la méthode **fs.open()**, ensuite nous décidons de lire un maximum de 128 bytes (ligne 8) en commençant à partir du début (position = 0). Le callback passé à la méthode **fs.read()** est appelé lorsque :

Fig.36

```
1  'use strict';
2
3  var p = require('path');
4
5  module.exports.run = () => {
6
7    p.normalize('/usr/bin/..//local/lib///..'); // /usr/local
8    p.join('/usr/bin/', '..', 'local/lib/', '..'); // /usr/local
9    p.relative('/sur/bin', '/sur/local/'); // ../local
10   p.resolve('/usr/', 'local', '../lib', '../local'); // /usr/local
11   p.extname('../app.js'); // .js
12   p.basename('../app.js'); // app.js
13   p.dirname('../app.js'); // ..
14
15 };
```

Fig.37

```
'use strict';

var fs = require('fs');

module.exports.run = () => {

  fs.stat('/etc/hosts', (error, stats) => {
    if(error){
      console.error(error.stack);
    }
    console.log(stats);
  });

};
```

Fig.38

```
wchegham 192.168.0.27  /tmp/programmez
$ ./run.sh
{ dev: 16777219,
  mode: 33188,
  nlink: 1,
  uid: 0,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 24908495,
  size: 1061,
  blocks: 8,
  atime: Sun Jul 12 2015 18:47:24 GMT+0200 (CEST),
  mtime: Wed May 13 2015 09:41:53 GMT+0200 (CEST),
  ctime: Wed May 13 2015 09:41:53 GMT+0200 (CEST),
  birthtime: Sun Jul 20 2014 19:30:07 GMT+0200 (CEST) }
wchegham 192.168.0.27  /tmp/programmez
$
```

Fig.39

```
1  'use strict';
2
3  var fs = require('fs');
4  module.exports.run = () => {
5
6    fs.open('./app.js', 'r', (err, fd) => {
7      if (err) { throw err }
8      var buffer = new Buffer(128),
9          offset = 0, length = buffer.length, position = 0;
10     var handleError = (err) => fs.close(fd, () => console.error('Closing file.', err));
11
12     fs.read(fd, buffer, offset, length, position, (err, bytes) => {
13       if (err) { handleError(err); }
14       console.log('just read ' + bytes + ' bytes');
15       if (bytes > 0) {
16         console.log(buffer.slice(0, bytes).toString());
17       }
18     });
19   });
20 };
```

- Une erreur se produit,
- Des données ont été lues,
- Rien n'a été lu. Fig 40

Une fois le fichier ouvert, puis lu, la méthode **fs.read()** a pu lire 79 bytes. Pour connaître le nombre exact de bytes à lire, nous pouvons utiliser la méthode **fs.fstat()** comme ceci : Fig 41 et 42

Nous pouvons donc nous baser sur l'attribut **size** de l'objet **fs.Stats** pour allouer un Buffer de la taille exacte à lire. Essayons maintenant d'écrire des données dans un fichier **./access.log** par exemple. Pour cela, nous allons utiliser la méthode **fs.write()** Fig 43 et 44

Dans cet exemple, nous avons ouvert le fichier **./access.log** en mode écriture "a" (avec concaténation). Ensuite, nous créons un Buffer avec un contenu textuel. À noter que la position est définie à NULL; ceci signifie que l'on veut écrire dans le fichier à la position courante du curseur. Parce que nous avons ouvert le fichier en mode "écriture concaténée", le curseur sera positionné à la fin du fichier. Pour ceux qui ont bien suivi les exemples précédents, vous remarquerez que nous n'avons pas fermé les fichiers ouverts. Pour des petits exemples comme ceux-là, L'OS peut s'arranger pour faire le ménage derrière nous en fermant ces fichiers lorsque le processus Node termine.

Mais pour des applications dans le vrai monde, il est important de fermer les fichiers lorsque vous avez fini de les manipuler.

Vous maîtrisez parfaitement l'API bas niveau de manipulation des fichiers avec Node. Sachez tout de même que Node nous offre d'autres méthodes assez "haut niveau" pour lire et écrire dans des fichiers sans se soucier ni de Buffer ni de descripteur. Ces méthodes sont **fs.readFile()** et **fs.writeFile()** ⊞

Remerciements

Je tiens à remercier tout particulièrement mes deux collègues Benoit Colombani et Jaffar Boudad pour leur patience et leurs relectures à la fois orthographique et technique.

La suite dans Programmez! 190.

```
2. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27 /tmp/programmez
$ ./run.sh
I read 79 bytes
'use strict';

var programmez = require('./lib/programmez');
programmez.run();

wchegham 192.168.0.27 /tmp/programmez
$
```

Fig.40

```
programmez.js — programmez
programmez.js
'use strict';

var fs = require('fs');
module.exports.run = () => {

  fs.open('./app.js', 'a', (err, fd) => {
    fs.fstat(fd, (error, stats) => {
      console.log(stats);
    });
  });

};
Line 1, Column 1; Saved ~/Documents/Articles/Programmez/node/programmez/lib/programmez.js (UTF-8)
```

Fig.41

```
2. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27 /tmp/programmez
$ ./run.sh
{ dev: 16777219,
  mode: 33188,
  nlink: 1,
  uid: 501,
  gid: 0,
  rdev: 0,
  blksize: 4096,
  ino: 59954596,
  size: 79,
  blocks: 8,
  atime: Tue Jul 14 2015 15:16:32 GMT+0200 (CEST),
  mtime: Tue Jul 14 2015 15:16:30 GMT+0200 (CEST),
  ctime: Tue Jul 14 2015 15:16:30 GMT+0200 (CEST),
  birthtime: Sun Jun 28 2015 19:34:16 GMT+0200 (CEST) }
wchegham 192.168.0.27 /tmp/programmez
$
```

Fig.42

```
programmez.js — programmez
programmez.js
'use strict';

var fs = require('fs');
module.exports.run = () => {

  fs.open('./access.log', 'a', (err, fd) => {
    if (err) { throw err; }
    var buffer = new Buffer('Programmez!\n'),
        bufferPosition = 0,
        length = buffer.length, position = null;
    fs.write( fd, buffer, bufferPosition, length, position, (err, bytes) => {
      if (err) { throw err; }
      console.log('I wrote ' + bytes + ' bytes');
    });
  });

};
Line 1, Column 1                                   Tab Size: 2    JavaScript
```

Fig.43

```
2. wchegham@wchegham-mbp-2: /tmp/programmez (zsh)
wchegham 192.168.0.27 /tmp/programmez
$ ./run.sh
I wrote 12 bytes
wchegham 192.168.0.27 /tmp/programmez
$ ./run.sh
I wrote 12 bytes
wchegham 192.168.0.27 /tmp/programmez
$ cat access.log

Programmez!
Programmez!
wchegham 192.168.0.27 /tmp/programmez
$
```

Fig.44

```
write.js — programmez
programmez.js
'use strict';
module.exports.run = () => require('./write.js');

write.js
'use strict';
var fs = require('fs');
fs.open('./access.log', 'a', (error, fd) => {

  if (error) { throw error; }
  var buffer = new Buffer('Programmez!'),
      bufferPosition = 0,
      length = buffer.length, position = null;

  fs.write( fd, buffer, bufferPosition, length, position, (err, bytes) => {

    if (err) { fs.close(fd, () => console.error('Closing file.', err)) }
    fs.close(fd, () => console.log('I wrote ' + bytes + ' bytes'));

  });
});
Line 13, Column 68                                 Tab Size: 2    JavaScript
```

Fig.45

Références

(1) http://arstechnica.com/information-technology/2013/05/native-level-performance-on-the-web-a-brief-examination-of-asm-js/3/
(2) https://github.com/creationix/nvm
(3) http://semver.org/
(4) https://en.wikipedia.org/wiki/Coupling_(computer_programming)
(5) https://dojotoolkit.org/
(6) http://requirejs.org/
(7) http://www.2ality.com/2014/09/es6-modules-final.html
(8) http://www.modulecounts.com/
(9) https://nodejs.org/docs/latest/api/modules.html#modules_caching
(10) https://github.com/mroderick/PubSubJS
(11) https://github.com/manekinekko/devoxx-2015-game-front/blob/master/app/scripts/Utils/PubSub.js
(12) https://fr.wikipedia.org/wiki/Syndrome_du_plat_de_spaghettis

Une gestion robuste et intuitive des dates enfin au rendez-vous avec Java 8 !

Les concepts de date et de temps sont fondamentaux dans la plupart des applications. Ultra présent en entreprise, Java proposait cependant un support inadéquat jusqu'alors pour la gestion des dates. Créée à l'origine de Java, l'API Date souffre de manques historiques liés à des erreurs de design qui rendent la vie impossible aux développeurs Java depuis de nombreuses années. Finalement, après une longue attente, la lumière arrive enfin au bout du tunnel avec Java 8 et son API Date and Time portée par la JSR-310. Tour du propriétaire d'une API amenée à améliorer la productivité des développeurs Java.



Sylvain Saurel
Ingénieur d'Etudes Java / JEE
sylvain.saurel@gmail.com

Représentation d'une date de naissance, d'une période de location, de l'heure d'un évènement ou bien encore des horaires d'ouverture d'un magasin, les cas d'utilisation pour les dates et les heures ne manquent pas et sont au coeur des applications d'entreprise. Pour représenter et manipuler ces différents concepts, l'API standard de Java ne proposait jusqu'à présent aucune solution réellement adaptée.

Un peu d'histoire

Pour mieux comprendre le problème, il est intéressant de revenir à ses racines. A la sortie de la version 1.0 de Java en 1996, un support des dates et du temps est proposé via la classe java.util.Date. Très rapidement, les développeurs se rendent compte que cette classe ne modélise pas une date à proprement parler. En réalité, il s'agit d'un point instantané dans le temps mesuré à partir de l'époque 01/01/1970 avec une précision en millisecondes. Néanmoins, à cause de la méthode toString(), qui renvoie une version de la date et de l'heure basée sur le fuseau horaire de la JVM appelante, de nombreux développeurs ont considéré, à tort, qu'il s'agissait d'une classe gérant la notion de fuseau horaire. A la sortie de la version 1.1 en 1997, il est déjà trop tard pour corriger la classe Date et les concepteurs du langage mettent en avant une nouvelle API centrée autour de la classe java.util.Calendar. Las, la classe Calendar se révèle aussi mal conçue que la classe Date. Leurs principaux problèmes sont les suivants :

- Mutable. Les classes représentant une date ou une heure se doivent d'être immutables.
- Offset. Les années dans la classe Date commencent en 1900, les mois dans les 2 classes commencent à 1 et les jours commencent à 0 ce qui n'est pas vraiment intuitif.
- Nommage. La classe Date ne représente pas vraiment une date et un Calendar n'est pas un calendrier au sens propre du terme.
- Formatage. Les formateurs fonctionnent uniquement avec la classe Date et ne proposent aucun support pour la classe Calendar.
- Thread-safe. Les classes existantes telles que java.util.Date et SimpleDateFormatter ne sont pas thread-safe ce qui peut conduire à des problèmes d'accès concurrentiels que le développeur moyen ne s'attend pas à devoir gérer.

Loin d'être exhaustive, cette liste de problématiques a conduit les développeurs à se tourner vers différentes alternatives au début des années 2000. Parmi celles-ci, Joda-Time, créée par Stephen Colebourne, s'est rapidement imposée comme la solution Java de référence pour une gestion de qualité des dates et du temps en Java.

Face à la popularité et l'omniprésence de Joda-Time en entreprise, il a finalement été décidé d'apporter un meilleur support pour la gestion des dates au sein du coeur du JDK. Ainsi, une nouvelle API a été créée à partir de zéro afin de s'affranchir des problématiques issues des erreurs passées. Attendue pour Java 7, son intégration a finalement été reportée pour la sortie de Java 8. Chapeauté sous l'égide de la JSR-310, le projet a été mené conjointement par Oracle et Stephen Colebourne, le père de Joda-Time, et placé au sein du nouveau package java.time apparu avec Java SE 8.

Vue d'ensemble

La nouvelle API Date and Time a été conçue autour de 3 idées centrales :

- Classes immutables. Une des principales faiblesses des formateurs de l'API Date originelle provenait du fait qu'ils n'étaient pas thread-safe. Ceci imposait aux développeurs de devoir les utiliser de manière thread-safe et avait tendance à complexifier leur développement quotidien.
- Conception liée au domaine. La nouvelle API modélise parfaitement son domaine avec des classes aux noms adéquats et représentant correctement les différents cas d'utilisation que ce soit pour les dates ou le temps. Finies les confusions liées à l'ancienne API qui proposait une classe Date pour manipuler un concept n'étant pas une date et une méthode toString() suggérant une gestion des fuseaux horaires alors que ceux-ci n'étaient pas gérés.
- Séparation des chronologies. La nouvelle API autorise les développeurs à travailler avec différents systèmes de calendriers afin de prendre en compte les besoins des utilisateurs dans le monde entier.

La nouvelle API s'articule autour de 5 packages :

- java.time qui est le package de base qui contient les objets de type valeur ;
- java.time.chrono qui donne accès aux différents systèmes de calendriers ;
- java.time.format pour le formatage et le parsing des dates et des heures ;
- java.time.temporal représentant le bas niveau du framework et qui permet d'étendre ses fonctionnalités ;
- java.time.zone fournissant les classes de support pour la gestion des fuseaux horaires.

Dates

Au coeur de l'API se trouve la classe LocalDate. Classe immutable de type valeur, son préfixe indique clairement que la date est locale du point de vue du contexte de l'observateur de la même manière qu'une horloge sur un mur. Afin d'éviter toute ambiguïté, elle n'est pas liée à des notions d'heure du jour ou de fuseau horaire. Ces notions sont de fait considérées à part. La classe LocalDate propose par ailleurs une API fluente avec toutes les méthodes facilitant sa création et la récupération d'informations :

```
LocalDate date = LocalDate.of(2014, Month.JUNE, 23);
int year = date.getYear(); // 2014
Month month = date.getMonth(); // JUIN
int dom = date.getDayOfMonth(); // 23
```

```
DayOfWeek dow = date.getDayOfWeek(); // Lundi
int len = date.lengthOfMonth(); // 30
boolean leap = date.isLeapYear(); // false (pas une année bissextile)
```

Avec la nouvelle API, tous les constructeurs sont désormais privés. De fait, la création d'objets tels que LocalDate se fait via des méthodes factories. On remarque également l'emploi d'enums pour modéliser les mois (classe Month) et les jours de la semaine (DayOfWeek) ce qui est plus lisible et fiable. LocalDate étant immutable, les opérations la concernant créent de nouvelles instances en sortie :

```
LocalDate date = LocalDate.of(2014, Month.JUNE, 23);
date = date.withYear(2015); // 2015-06-23
date = date.plusMonths(2); // 2015-08-23
date = date.minusDays(1); // 2015-08-22
```

Bien souvent, les opérations à réaliser s'avèrent plus complexes. Pour adresser ces cas, la nouvelle API propose le concept de TemporalAdjuster. Concrètement, un ajusteur est un bloc de code encapsulant des opérations logiques communes. On peut ainsi imaginer un PlusAdjuster utilisé pour ajouter ou soustraire des valeurs de certains champs sur une date locale existante. En standard, le JDK propose un certain nombre d'ajusteurs permettant par exemple d'obtenir l'instance de LocalDate correspondant au dernier jour du mois. L'avantage de l'API est qu'elle permet de créer ses propres ajusteurs pour les cas non couverts en standard. Utiliser un ajusteur est relativement aisé :

```
import static java.time.DayOfWeek.*
import static java.time.temporal.TemporalAdjusters.*

LocalDate date = LocalDate.of(2014, Month.JUNE, 23);
date = date.with(lastDayOfMonth());
date = date.with(nextOrSame(WEDNESDAY));
```

On remarque ici l'utilisation d'imports statiques rendant le code encore plus lisible et simple à écrire.

Dates et Heures en tant que valeurs

Avant d'aller plus loin dans la nouvelle API Date and Time, il est intéressant de revenir sur ce qu'implique le fait que la classe LocalDate soit de type valeur. Une valeur est ainsi un type de données pour lequel 2 instances égales sont entièrement interchangeables. L'exemple le plus évident étant la classe String pour laquelle on s'intéresse à l'égalité en utilisant equals() et non l'égalité au niveau référence via l'opérateur == . De fait, les classes de date et d'heure doivent être des valeurs et il n'y a aucune raison de comparer 2 instances de LocalDate via l'opérateur == . Il s'avère essentiel de bien comprendre ce principe pour maîtriser la nouvelle API.

Systèmes de calendriers alternatifs

A l'instar de LocalDate, la majorité des classes liées aux dates et aux heures présentes dans java.time repose sur un seul et unique système de calendrier défini par le standard ISO-8601. Connu sous le nom de calendrier Grégorien; ce système de calendrier est le standard de facto pour le monde civil. L'impact sur l'API Date and Time est la nécessité de disposer d'un ensemble de classes pour gérer d'autres systèmes de calendriers. L'interface Chronology est le point d'entrée principal pour les systèmes alternatifs. Par défaut, Java SE 8 propose 4 systèmes de calendriers supplémentaires : Bouddhiste, Minguo, Japonais et Musulman. D'autres systèmes pouvant évidemment être implémentés.

Chaque système de calendrier vient avec une classe date dédiée, ce qui correspond aux classes ThaiBuddhistDate, MinguoDate, JapaneseDate et HijrahDate. Celles-ci sont à utiliser uniquement dans le cas d'applications localisées pour une seule zone donnée. On peut ainsi imaginer utiliser JapaneseDate pour une application dédiée exclusivement au gouvernement Japonais. L'interface ChronoLocalDate sert de base d'abstraction pour ces classes, et LocalDate, autorise l'écriture de code sans connaître le système de calendrier sous-jacent. Sur le même plan, les interfaces ChronoLocalDateTime et ChronoZonedDateTime sont fournies. En pratique, les cas d'utilisation resteront très limités et s'avèreront complexes puisqu'il faudra s'assurer qu'aucune ligne de code ne fait de prérequis sur un système de calendrier en particulier.

Heure du jour

Une fois le concept des dates locales assimilé, on peut considérer l'heure du jour locale qui est modélisée au sein de la classe LocalTime. Le cas d'usage classique est l'heure d'ouverture et celle de fermeture d'un magasin. En considérant une chaîne de magasins aux Etats-Unis ouverte de 8h à 20h, on souhaite représenter une heure du jour locale sans notion de fuseau horaire puisque les horaires seront les mêmes pour chaque magasin localement. La classe LocalTime est bien entendu de type valeur et n'a aucun lien avec une date ou un fuseau horaire. Ajouter ou soustraire du temps à une heure du jour basculera autour de minuit. Ainsi, ajouter 2h à 23h donnera une instance de LocalTime valorisée à 1h. Les exemples sont globalement identiques à ceux de LocalDate :

```
LocalTime time = LocalTime.of(21, 45);
int hour = date.getHour(); // 21
int minute = date.getMinute(); // 45
time = time.withSecond(15); // 21:45:15
time = time.plusMinutes(3); // 21:48:15
```

Le mécanisme des ajusteurs est également applicable aux instances de LocalTime.

Combiner Date et Heure

Certains cas d'utilisation vont nécessiter de considérer la date et l'heure du jour. Combinant LocalDate et LocalTime, la classe LocalDateTime est chargée de modéliser ce concept. Ici encore, la date et l'heure sont représentées sans notion de fuseau horaire. La création d'une instance de LocalDateTime peut se faire directement ou bien en combinant des instances de date et d'heure :

```
LocalDateTime dt1 = LocalDateTime.of(2014, Month.JUNE, 10, 20, 30);
LocalDateTime dt2 = LocalDateTime.of(date, time);
LocalDateTime dt3 = date.atTime(20, 30);
LocalDateTime dt4 = date.atTime(time);
```

La classe LocalDateTime propose également un support des ajusteurs :

```
import static java.time.temporal.TemporalAdjusters.*;

LocalDateTime timePoint = ...
foo = timePoint.with(lastDayOfMonth());
bar = timePoint.with(previousOrSame(ChronoUnit.WEDNESDAY));
timePoint.with(LocalTime.now());
```

Conçue de la même manière, les classes LocalDate, LocalTime et LocalDateTime proposent les mêmes modèles de méthodes pour les opérations qu'elles supportent. De fait, l'apprentissage de l'API s'en trouve accéléré. On retrouve ainsi des méthodes factories telles que of, from, now ou parse mais également des méthodes de manipulation comme get, with, plus ou minus et enfin des méthodes utilitaires comme to, at ou is.

Instant

Lorsqu'il manipule des dates et des heures, l'homme raisonne avant tout en termes d'années, mois, jours, heures, minutes et secondes. Ce modèle de temps n'est évidemment pas unique et constitue la vision humaine. Un autre modèle de temps que l'on peut considérer est celui lié à la machine. Dans cette vision, un point instantané sur la ligne de temps est un simple nombre. Il s'agit de l'approche la plus simple à implémenter au niveau des ordinateurs et c'est donc logique qu'elle soit adoptée par les systèmes UNIX par exemple. Ces derniers comptent le temps en secondes à partir du 01/01/1970 ce qui est implémenté en Java par le nombre de millisecondes depuis 1970. L'API java.time propose une vision machine du temps au travers de la classe valeur Instant. Cette dernière modélise un instant de la ligne de temps qui n'est lié à aucun contexte :

```
Instant start = Instant.now();
// ...
Instant end = Instant.now();
// test
if (end.isAfter(start)) {
  // ...
}
```

L'usage le plus fréquent de la classe Instant concerne le stockage et la comparaison de timestamps représentant des instants d'évènements non liés à des fuseaux horaires. L'appel de méthodes comme get ou plus ne pourra de fait être satisfait sans la précision d'un fuseau horaire et lancera une exception. En effet, d'un point de vue machine, ces informations n'ont pas de signification réelle.

Fuseaux Horaires

Inventé au Royaume-Uni, le concept de fuseau horaire sert à lier une date et une heure à un contexte. La définition des différents fuseaux résultant de choix politiques, il sont amenés à changer au fil du temps. Avec l'ancienne API de manipulation des dates, la classe TimeZone était mise à disposition pour modéliser le concept de fuseau horaire. Désormais, la nouvelle API se base sur la classe ZoneId qui présente 2 différences majeures avec TimeZone. En premier lieu, c'est une classe immutable. Enfin, les règles liées aux fuseaux sont définies au sein de la classe déléguée ZoneRules et non directement au sein de ZoneId. Un simple appel à la méthode getRules() de ZoneId permettant d'obtenir les règles. Un cas d'utilisation classique des fuseaux consiste à définir un offset par rapport au fuseau référence UTC / Greenwich. La classe ZoneOffset, qui hérite de ZoneId, modélise l'offset au niveau heure avec la méridien zéro de Greenwich à Londres. En tant que développeur, il est préférable d'éviter au maximum de devoir gérer les fuseaux horaires et donc de recourir en priorité aux classes locales. Néanmoins, si c'est impossible, il faudra recourir à ZonedDateTime qui gère la conversion entre la vision humaine du temps et la vision machine. La création d'une date liée à un fuseau se faisant comme suit :

```
ZoneId zone = ZoneId.of("Europe/Paris");

LocalDate date = LocalDate.of(2014, Month.JUNE, 10);
ZonedDateTime zdt1 = date.atStartOfDay(zone);

Instant instant = Instant.now();
ZonedDateTime zdt2 = instant.atZone(zone);
```

Ici, on remarque l'appel à la méthode atZone de la classe Instant pour obtenir une classe ZonedDateTime équivalente à l'Instant mais avec une notion de fuseau horaire en plus. Les développeurs ayant déjà eu affaire aux fuseaux horaires savent pertinemment que la gestion de l'heure d'été, connue sous l'abréviation DST (Daylight Saving Time), est une problématique importante. Avec l'heure d'été, l'offset par rapport au fuseau référence peut être changé au moins 2 fois par an. For heureusement, la nouvelle API gère ces ajustements sous la forme d'offsets de transition implémentés via la classe ZoneOffsetTransition.

Durées et Périodes de temps

Les classes relatives aux dates et aux heures présentées jusqu'ici représentent des points sur la ligne de temps. Additionner 2 valeurs de ces types permet d'obtenir une quantité de temps. La classe Duration modélise une quantité de temps mesurée en secondes et nanosecondes. Pour représenter une quantité de temps mesurée en années, mois et jours, telle que 3 ans 8 mois et 13 jours, la classe Period est à disposition du développeur Java. Ces quantités permettent de réaliser des opérations sur les dates et les heures :

```
Period period = Period.of(3, 8, 13);

LocalDate newDate = oldDate.plus(period);
ZonedDateTime newDateTime = oldDateTime.minus(period);

Period sixMonths = Period.ofMonths(6);
LocalDate date = LocalDate.now();
LocalDate future = date.plus(sixMonths);

Duration duration = Duration.ofSeconds(45, 9);
Duration oneDay = Duration.between(today, yesterday);
```

Formatage et parsing

Formater une date suivant un modèle ou obtenir une instance de date à partir d'une chaîne de caractères sont des opérations basiques. Le package java.time.format de l'API est dédié à cet effet et s'articule autour de la classe DateTimeFormatter et son builder associé DateTimeFormatterBuilder. Les formateurs sont créés via l'emploi de méthodes statiques exposées par DateTimeFormatter :

```
DateTimeFormatter f = DateTimeFormatter.ofPattern("dd/MM/uuuu");
LocalDate date = LocalDate.parse("24/06/2014", f);
String str = date.format(f);
```

Contrairement à l'ancienne API, l'API Date and Time propose des méthodes acceptant en entrée le système de calendrier utilisé ou le fuseau horaire considéré. Enfin, il reste possible d'obtenir un contrôle plus bas niveau sur le travail réalisé par DateTimeFormatter pour des cas d'usage plus complexes.

Conclusion

Attendue depuis longtemps, la nouvelle API Date and Time amenée par Java 8 fournit au développeur un modèle clair et sûr pour la manipulation des dates et du temps dans les applications Java. Se basant sur le travail réalisé au sein de Joda-Time, l'API va encore plus loin et permettra aux développeurs de dire définitivement adieu aux classes java.util.Date et Calendar, sources de tant de bugs. Cet abandon définitif sera facilité par le support au sein du JDK de cette nouvelle API par les autres APIs telles que JDBC par exemple. Modélisant parfaitement son domaine, avec une couverture d'un vaste éventail de cas d'utilisation, l'API Date and Time permettra au développeur Java d'apprécier à nouveau la manipulation des dates et du temps !



Le Bluetooth Low Energy dans les applications universelles

3e partie

Dans la 3ème et dernière partie de cet article, nous allons voir comment gérer le capteur de pression atmosphérique dont le paramétrage est un peu différent des autres capteurs du SensorTag, et voir comment afficher toutes les valeurs connectées dans une interface fonctionnant à la fois sur PC et Windows Phone.



Stéphane Sibué
Directeur technique chez GUYZMO
stephane.sibue@guyzmo.fr - www.guyzmo.fr



Le capteur de pression atmosphérique est un peu différent

Tous les capteurs du SensorTag fonctionnent de la même manière, à part celui de pression qui a besoin d'une phase de calibration. Sinon pour le reste, c'est par exemple la même chose qu'avec le capteur de température.

BarometerSensor

Le **BarometerSensor** hérite comme les autres de **SensorBase** et comme pour les autres, il lui faut une propriété permettant de lire la valeur actuelle de ce capteur, et un constructeur correctement paramétré :

```
protected double pCurrentPressure;

public double CurrentPressure
{
  get { return pCurrentPressure; }

  private set
  {
    pCurrentPressure = value;
    NotifyPropertyChanged("CurrentPressure");
  }
}

public BarometerSensor()
: base("F000AA40-0451-4000-B000-000000000000",
"F000AA42-0451-4000-B000-000000000000",
"F000AA41-0451-4000-B000-000000000000")
{

}
```

Il lui faut aussi accéder à une caractéristique particulière permettant de récupérer un tableau de 16 octets (8 mots de 16 bits en fait) qui contiendra les valeurs de calibration. Ces valeurs seront utilisées dans la formule de calcul de la pression atmosphérique.
On a donc besoin de l'UUID de la caractéristique de calibration et d'un tableau dans lequel les valeurs de calibration seront récupérées :

```
protected int[] pBarometerCalibrationData = new int[] { 0, 0, 0, 0, 0, 0, 0, 0 };
protected Guid BAROMETER_CALIBRATION_UUID = new Guid("F000AA43-0451-4000-B000-000000000000");
```

Il faut ensuite surcharger la méthode d'initialisation pour que l'opération de calibration soit prise en charge à ce moment-là. Pour lire les données de calibration, il suffit d'écrire la valeur numérique 2 dans la caractéristique de configuration, et d'aller ensuite lire les données mises à disposition dans la caractéristique de calibration. Il ne faut pas oublier en premier lieu d'appeler la méthode **Init** du parent :

```
public override async System.Threading.Tasks.Task<SensorBase.InitResult> Init()
{
  var r = await base.Init();

  if (r == InitResult.Ok)
  {
    // Lecture des données de calibration

    GattCommunicationStatus s;

    using (var writer = new DataWriter())
    {
      writer.WriteByte(2);
      s = await pConfigurationCharacteristic.WriteValueAsync(writer.DetachBuffer());
    }

    if (s == GattCommunicationStatus.Success)
    {
      var calib = GetCharacteristic(pDeviceService, BAROMETER_CALIBRATION_UUID);

      if (calib != null)
      {
        var result = await calib.ReadValueAsync(BluetoothCacheMode.Uncached);

        if (result.Status == GattCommunicationStatus.Success && result.Value.Length == 16)
        {
          byte[] b = new byte[16];

          using (var wReader = DataReader.FromBuffer(result.Value))
          {
            wReader.ReadBytes(b);
          }

          pBarometerCalibrationData[0] = BitConverter.ToUInt16(b, 0);
          pBarometerCalibrationData[1] = BitConverter.ToUInt16(b, 2);
          pBarometerCalibrationData[2] = BitConverter.ToUInt16(b, 4);
          pBarometerCalibrationData[3] = BitConverter.ToUInt16(b, 6);
          pBarometerCalibrationData[4] = BitConverter.ToInt16(b, 8);
          pBarometerCalibrationData[5] = BitConverter.ToInt16(b, 10);
          pBarometerCalibrationData[6] = BitConverter.ToInt16(b, 12);
          pBarometerCalibrationData[7] = BitConverter.ToInt16(b, 14);
        }
      }
    }
```

```
    }

    return r;
}
```

Nous avons presque terminé, il ne reste plus qu'à coder la partie qui calcule la pression courante :

```
protected override void OnValueChanged(IBuffer buffer)
{
    int t_r, p_r;
    double t_a, S, O;

    using (var wReader = DataReader.FromBuffer(buffer))
    {
        byte[] b = new byte[4];
        wReader.ReadBytes(b);

        t_r = BitConverter.ToInt16(b, 0);
        p_r = BitConverter.ToUInt16(b, 2);
    }

    t_a = (100 * (pBarometerCalibrationData[0] * t_r / Math.Pow(2, 8) + pBarometerCalibration
Data[1] * Math.Pow(2, 6))) / Math.Pow(2, 16);
    S = pBarometerCalibrationData[2] + pBarometerCalibrationData[3] * t_r / Math.Pow(2, 17)
 + ((pBarometerCalibrationData[4] * t_r / Math.Pow(2, 15)) * t_r) / Math.Pow(2, 19);
    O = pBarometerCalibrationData[5] * Math.Pow(2, 14) + pBarometerCalibrationData[6]
 * t_r / Math.Pow(2, 3) + ((pBarometerCalibrationData[7] * t_r / Math.Pow(2, 15)) * t_r)
 / Math.Pow(2, 4);
    CurrentPressure = (S * p_r + O) / Math.Pow(2, 14);
}
```

Et voilà, un capteur de pression prêt à être utilisé !

La page d'affichage

Comme la page est en portrait sous Windows Phone et en paysage sous Windows, j'ai décidé, pour ne pas m'embêter, de garder pour les 2 projets une **MainPage** différente, c'est la seule chose qui n'est pas dans le projet Shared en fait **Fig.1**. Les 3 valeurs à afficher (pression atmosphérique, température et humidité) sont bindées directement aux valeurs fournies par les capteurs correspondants. Les 3 valeurs brutes sont affichées dans des **TextBlock** :

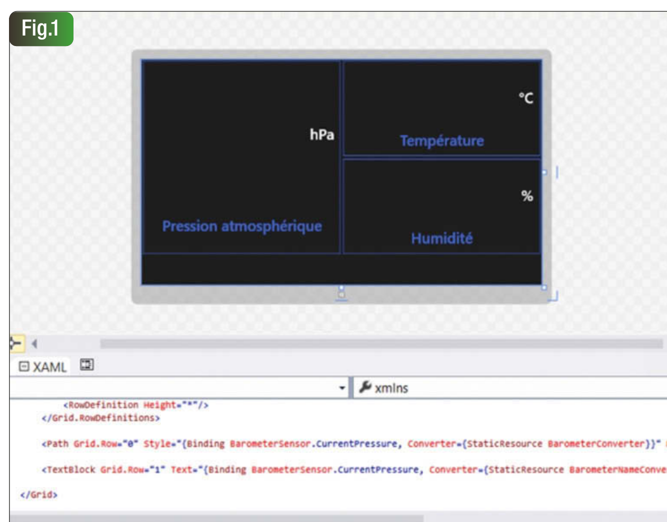


Fig.1

```
<TextBlock Text="{Binding BarometerSensor.CurrentPressure, Converter={StaticResource
 PressureConverter}}" VerticalAlignment="Center" HorizontalAlignment="Center" Style=
"{StaticResource TitleTextBlockStyle}" FontSize="120" />

<TextBlock Text="{Binding TemperatureSensor.CurrentTemperature, Converter={Static
Resource TemperatureConverter}}" VerticalAlignment="Center" HorizontalAlignment
="Center" Style="{StaticResource TitleTextBlockStyle}" FontSize="120" />

<TextBlock Text="{Binding HumiditySensor.CurrentHumidity, Converter={StaticResource
 HumidityConverter}}" VerticalAlignment="Center" HorizontalAlignment="Center" Style=
"{StaticResource TitleTextBlockStyle}" FontSize="120" />
```

La pression est exprimée en Pa (Pascal) et elle est affichée en hPA (hecto Pascal); on utilise un convertisseur pour effectuer cette transformation. Même chose pour la température et l'humidité qu'on ne souhaite afficher qu'avec un seul chiffre après la virgule. Voici par exemple le code du convertisseur pour la pression atmosphérique :

```
public class PressureConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, string language)
    {
        // On récupère un double correspondant à une valeur en Pa (pascal)
        // On la divise par 100 pour obtenir des hPa
        // On retourne le résultat sans aucun chiffre après la virgule

        if (value is double)
        {
            double v = (double)value / 100.0;

            if (double.IsNaN(v))
            {
                return "?";
            }
            else
            {
                return v.ToString("0");
            }
        }
        else
        {
            return null;
        }
    }

    public object ConvertBack(object value, Type targetType, object parameter, string language)
    {
        throw new NotImplementedException();
    }
}
```

Le ViewModel de la MainPage

Dans le modèle **MVVM** les données ne sont pas directement manipulées par le code de l'interface mais par un **ViewModel**. **MainPageViewModel** est chargé de cette tâche. C'est lui qui instancie les capteurs, les initialise et les rend accessibles au moteur de binding. Voici le code d'initialisation des capteurs :

```
public TemperatureSensor TemperatureSensor { get; private set; }
public BarometerSensor BarometerSensor { get; private set; }
```

```
public HumiditySensor HumiditySensor { get; private set; }

private bool pSensorTagConnected;

public bool SensorTagConnected
{
  get { return pSensorTagConnected; }

  set
  {
     pSensorTagConnected = value;
     NotifyPropertyChanged("SensorTagConnected");
  }
}

public async Task<bool> Init()
{
  // On crée les 3 capteurs et on les active

  TemperatureSensor = new TemperatureSensor();
  HumiditySensor = new HumiditySensor();
  BarometerSensor = new BarometerSensor();

  var temperatureResult = await TemperatureSensor.Init();
  var humidityResult = await HumiditySensor.Init();
  var barometerResult = await BarometerSensor.Init();

  if (temperatureResult == SensorBase.InitResult.Ok &&
   humidityResult == SensorBase.InitResult.Ok &&
   barometerResult == SensorBase.InitResult.Ok)
  {
     await TemperatureSensor.StartSensor();
     await HumiditySensor.StartSensor();
     await BarometerSensor.StartSensor();

     SensorTagConnected = true;
  }

  return SensorTagConnected;
}
```

A partir de là, les données sont directement affichées à l'écran dès qu'elles changent dans le SensorTag, sans aucune intervention particulière, grâce au système de notifications du BLE et au binding XAML **Fig.2**. Pour ce qui est de l'affichage du picto lié à la pression atmosphérique, ou la mise en œuvre détaillée du ViewModel de MainPage, je vous laisse regarder en détails les sources de l'application que vous trouverez ici : http://1drv.ms/1EhRWfw

Conclusion

Le développement d'applications universelles pour Windows et Windows Phone utilisant le Bluetooth Low Energy est rendu extrêmement simple grâce à une approche claire, simple et efficace. Il est ainsi possible de réaliser, en peu de temps, des belles applications capables de dialoguer avec une multitude d'objets connectés.
Il ne reste plus qu'à espérer que la limitation actuelle ne permettant pas « d'accrocher » à la volée un appareil BLE ait disparue dans Windows Phone 10 et Windows 10, ce qui permettra de réaliser les mêmes types d'applications qu'avec iOS ou Android.

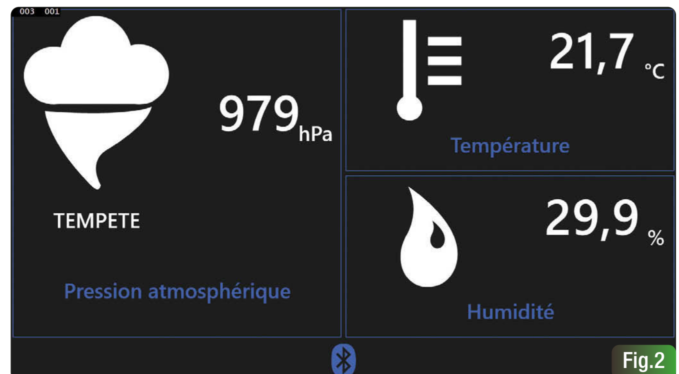


Fig.2

ASP.NET 5 et Docker dans Microsoft Azure

Cet article a pour but de vous présenter comment il est possible d'héberger une application développée avec ASP.NET 5 dans un conteneur Docker sur Azure.



Julien Corioland - @jcorioland
Microsoft Azure Technical Evangelist
Microsoft France

Quelques mots sur ASP.NET 5

ASP.NET 5 est la prochaine version d'ASP.NET, actuellement en cours de développement et dont la version 1.0 est prévue pour le premier trimestre 2016 (à l'écriture de ces lignes, nous utilisons la Beta 6).

Ce nouveau Framework fait l'objet d'une réécriture complète, contrairement aux dernières évolutions que nous avons pu connaître ces dernières années, et ce, dans le but de proposer un Framework plus flexible, plus robuste, plus léger et surtout plus adapté aux problématiques du développement Web d'aujourd'hui et aux plateformes d'hébergement modernes, telles que Microsoft Azure, par exemple.

Avec ASP.NET 5, on assiste également à la réunification en un seul et même Framework de MVC, Web API et prochainement Web Pages, tous englobés sous le nom d'ASP.NET MVC 6. Une autre nouveauté que l'on se doit de mettre en avant est la capacité des applications ASP.NET 5 à être exécutées nativement sur Linux et sur OS X, à l'aide de la Core CLR, un nouveau moteur d'exécution plus léger que la CLR que nous connaissons et qui sera, à terme, disponible sur Windows, Linux et OS X.

Cela signifie qu'il sera possible d'héberger une application Web développée en C#, avec ASP.NET 5, sur une machine non Windows, en s'affranchissant de Mono !

Docker et Microsoft, main dans la main !

Cela ne vous a certainement pas échappé, depuis plusieurs mois, Microsoft et Docker travaillent main dans la main pour fournir la meilleure expérience possible dans l'utilisation de Docker dans Microsoft Azure, actuellement sous Linux, et dans un futur assez proche sous Windows, puisque Microsoft a annoncé lors de sa conférence //Build que les conteneurs Windows Server et Hyper-V feraient leur apparition avec Windows Server 2016, et que c'était la technologie Docker qui avait été choisie pour les piloter !

Aujourd'hui, Microsoft fournit une extension de machine virtuelle qui permet de configurer un hôte Docker Linux dans Azure en quelques clics depuis le portail, ou en ligne de commande via l'outil Azure Cross-Platform CLI (disponible via NPM). Nous verrons en détail ce point par la suite.

Sont également supportés des scénarios plus avancés tels que Docker Swarm ou Docker Compose ou même la création d'un « Docker Trusted Registry » hébergé dans Azure, pour y stocker vos images privées. Récemment, une catégorie « Container Apps » a même fait son apparition dans le portail Microsoft Azure : Fig.1.

Depuis le 20 Août 2015, nous avons également accès, en « preview » à une première version des conteneurs Windows Server 2016 (dans Azure ou on-premise) que l'on peut piloter depuis Docker ou Powershell. Egalement disponibles en preview, des outils de publication vers un hôte Docker (Linux ou Windows) depuis Visual Studio 2015, en quelques clics.

Pourquoi Docker ?

La question que l'on peut se poser, et qui est tout à fait légitime, est pourquoi utiliser Docker pour héberger une application ASP.NET 5 (ou autre d'ailleurs), que cela soit dans un conteneur Linux ou dans le futur dans un conteneur Windows Server 2016 ?

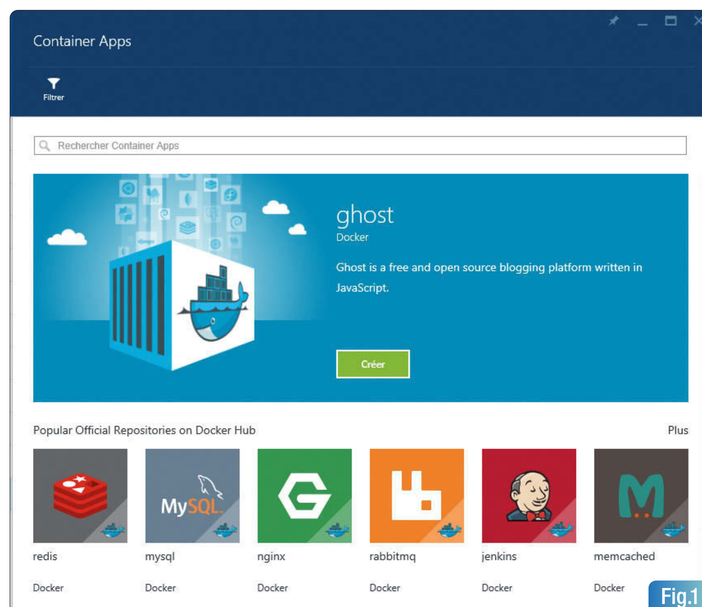


Fig.1

Les conteneurs, couplés à la technologie Docker et les outils qui vont avec (images, Dockerfile...) permettent de gérer très simplement le déploiement d'un applicatif sur différents environnements (développement, recette, QA, production...), par exemple sans avoir à réinstaller la machine ou booter une machine virtuelle, à partir d'une image. Dès lors que vous avez un hôte Docker qui tourne, le déploiement d'une application peut se faire en quelques secondes.

Un autre avantage est que Docker ne fait pas de la virtualisation de machine, mais de la virtualisation d'OS, ce qui implique que, théoriquement, un conteneur Docker qui tournerait chez tel ou tel fournisseur pourrait être déployé chez un autre en quelques minutes seulement (pas d'images de VM à refaire, pas de disques à migrer etc...).

Ce genre de technologies va permettre de simplifier un certain nombre de scénarios DevOps, puisque la définition de l'environnement d'exécution (le conteneur, instancié à partir de son image) d'une application est la même du poste du développeur jusqu'à la production.

Enfin, on peut souligner aussi le fait qu'avec Docker tout a été pensé pour être scriptable et versionnable, ce qui permet de stocker dans le même dépôt (Visual Studio Online, Git ou autre...) le code source de l'application mais aussi la définition de son environnement d'exécution, pour une version bien déterminée. Il devient alors très simple de revenir à une version antérieure pour investiguer et corriger un bug, par exemple.

Le contexte étant posé, je vous propose de rentrer dans le vif du sujet, et de commencer par la création d'un hôte Docker dans Azure.

Création d'un hôte Docker dans Azure

Pour continuer, vous devez être en possession d'un compte sur la plateforme Microsoft Azure. Si vous n'en n'avez pas et que vous souhaitez tester l'offre, vous avez la possibilité de créer un compte d'essai, valide un mois, depuis cette page : https://azure.microsoft.com/fr-fr/pricing/free-trial/.

NB : Si vous êtes sur un poste Windows, vous devez également installer Git avec l'intégration des outils Unix dans CMD et Bash Git. Vous trouverez ces outils sur cette page : https://git-for-windows.github.io/

Tout d'abord, vous allez devoir installer les Microsoft Azure Cross-platform

Command Line Tools. Si vous êtes sous Windows, vous pouvez le faire à l'aide de Web Platform Installer : Fig.2.
Si vous êtes sous Linux ou OS X, vous pouvez installer les outils via npm à l'aide de la commande suivante :

```
> npm install azure-cli -g
```

Une fois les outils installés, vous allez devoir vous connecter à votre compte Azure, tout d'abord en téléchargeant un fichier « .publishsettings » depuis le portail, à l'aide de la commande :

```
> azure account download
```

Puis en important ce fichier, à l'aide de la commande

```
> azure account import [chemin_fichier_publishsettings]
```

NB : *si vous utilisez un compte d'entreprise pour vous connecter à Azure, vous pouvez utiliser la commande « azure login » et vous connecter directement avec votre email / mot de passe.*

Vous pouvez ensuite valider la bonne importation de votre compte Azure à l'aide de la commande :

```
> azure account show
```

Fig.3.
Utilisez la commande « azure vm image list » pour trouver l'image Ubuntu 14.04 LTS à utiliser, de la manière suivante :

```
> azure vm image list | grep 14_04
```

Copiez l'identifiant de la dernière image stable en date, au moment de l'écriture de cet article, il s'agit de « b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04_3-LTS-amd64-server-20150805-en-us-30GB »
Vous pouvez alors utiliser la commande suivante pour demander la création d'une instance de cette image :

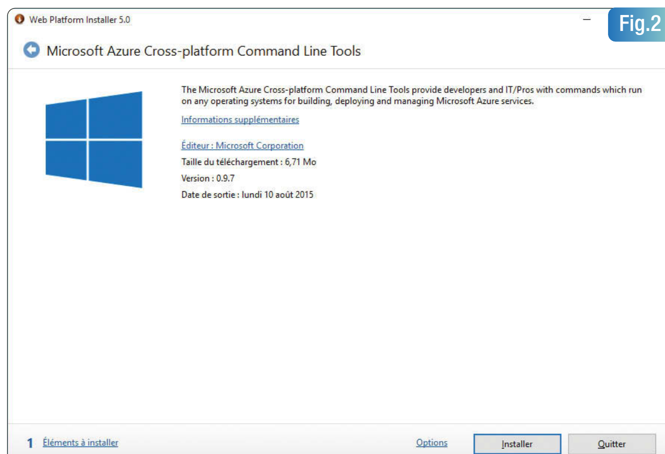


Fig.2

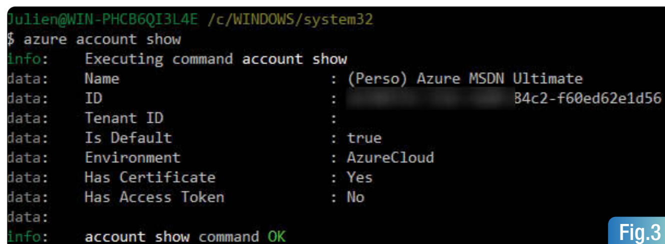


Fig.3

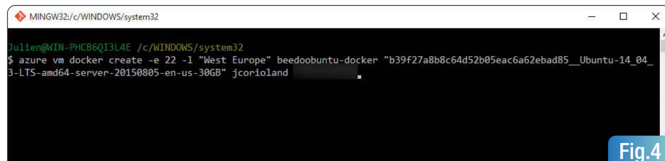


Fig.4

```
> azure vm docker create -e 22 -l "West Europe" <nom-vm> "b39f27a8b8c64d52b05eac6a62ebad85__Ubuntu-14_04_3-LTS-amd64-server-20150805-en-us-30GB" <username> <password>
```

Avec :
- 22 : le port sur lequel vous souhaitez vous connecter en SSH à la machine ;
- West Europe : le data center dans lequel vous souhaitez créer la machine ;
- <nom-vm> : le nom que vous souhaitez donner à la machine ;
- <username> : le nom d'utilisateur qui vous permettra de vous connecter en SSH ;
- <password> : le mot de passe qui vous permettra de vous connecter en SSH ;

Fig.4.
Patientez jusqu'à ce que la VM soit créée, puis connectez-vous à celle-ci à l'aide de votre client SSH favori, en utilisant comme adresse nom-vm.cloudapp.net (en remplaçant nom-vm par le nom que vous avez choisi à la création), sur le port 22 et avec le nom d'utilisateur et mot de passe que vous avez choisi. Une fois connecté, vous pouvez vérifier que Docker a bien été automatiquement installé sur la machine, en tapant la commande :

```
> docker --version
```

Fig.5.
NB : *l'activation de l'extension docker dans la machine virtuelle peut prendre un peu de temps, soyez patient...*
Vous pouvez également obtenir des informations sur l'hôte Docker en tapant la commande :

```
> docker info
```

Fig.6.

Votre hôte Docker est prêt !

NB : *pour l'accès distant à un hôte Docker depuis votre poste (avec le client Docker déjà installé), les outils Azure Cross-Platform CLI ont automatiquement généré des certificats qui ont été placés dans le répertoire .docker, dans le répertoire racine de votre compte utilisateur. Pour simplifier l'écriture des commandes dans cet article, nous avons choisi de nous connecter directement en SSH sur la machine hôte Docker.*

Récupération du projet ASP.NET 5

Pour simplifier la lecture de cet article et ne pas le surcharger avec trop de code, un projet d'exemple ASP.NET 5 est récupérable sur GitHub : https://github.com/jcorioland/AspNet5Programmez.git.
Placez-vous dans le dossier de votre choix et exécutez la commande suivante :

```
> git clone https://github.com/jcorioland/AspNet5Programmez.git
```

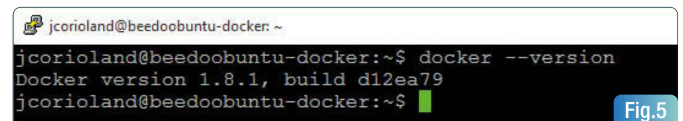


Fig.5

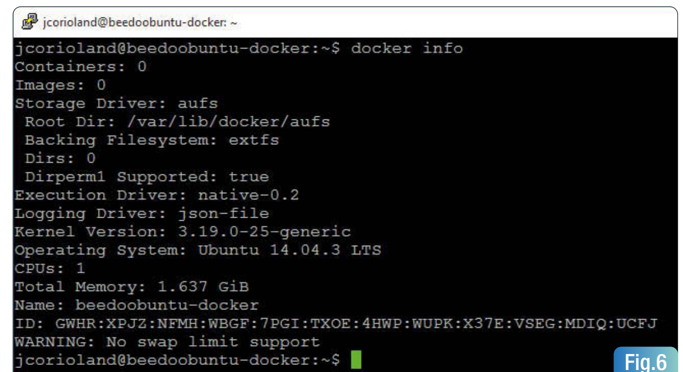


Fig.6

Naviguez ensuite dans le dossier AspNet5Programmez/src. En listant le contenu, vous verrez que ce projet contient deux éléments remarquables :

- Le dossier AspNetDockerProgrammez, qui contient le code source de l'application, et notamment le fichier project.json, qui liste les différentes dépendances ;
- Le fichier Dockerfile, qui nous intéresse plus particulièrement ici, et qui va nous permettre de créer une image Docker pour pouvoir instancier l'application dans un conteneur.

Comme vous pouvez le constater, le fichier Dockerfile ne contient que très peu de contenu :

```
# Utilisation de l'image de base aspnet 5 beta 5
FROM microsoft/aspnet:1.0.0-beta6

MAINTAINER Julien CORIOLAND

# On créé le dossier qui contiendra les sources de l'application
RUN mkdir -p /opt/src

# On se place dans le dossier /opt/src
WORKDIR /opt/src

# On ajoute le code de l'application au dossier courant
COPY ./AspNetDockerProgrammez /opt/src

# Exécution de la restauration des packages NuGet
RUN dnu restore

# Exposition du port 5004 utilisé par le serveur Kestrel
EXPOSE 5004

# Démarrage de l'application en utilisant DNX
ENTRYPOINT ["dnx", ".", "kestrel"]
```

Tout d'abord, nous indiquons que nous souhaitons nous baser sur l'image « microsoft/aspnet » en version beta-1.0.0.6, proposé par Microsoft dans le hub communautaire de Docker (http://hub.docker.com), et ce à l'aide de la directive **FROM**.

La directive **RUN** permet d'exécuter des commandes à l'intérieur du conteneur. Nous l'utilisons pour créer un répertoire qui contiendra les sources de l'application.

La directive **WORKDIR** nous permet de définir le dossier courant dans le conteneur.

La directive **COPY** permet de copier un contenu local à l'intérieur du conteneur : ici, nous copions le code source de l'application dans le répertoire /opt/src du conteneur Docker.

Nous utilisons à nouveau la commande **RUN** pour lancer la restauration des packages NuGet, à l'aide de l'outil DNU (.NET Development Utilities).

La directive **EXPOSE** permet d'indiquer que le conteneur expose le port 5004 à la machine hôte. Enfin, nous indiquons que le point d'entrée du conteneur est l'exécution de la commande DNX dans le répertoire courant « . » avec le paramètre « kestrel », et ce à l'aide de la directive **ENTRYPOINT**.

Création de l'image Docker

C'est à partir du Dockerfile que nous allons pouvoir générer une image Docker, qui nous permettra ensuite d'instancier un ou plusieurs conteneurs exécutant l'application.

Pour générer l'image Docker, on utilise la commande suivante :

```
> docker build -t "aspnet5programmez" .
```

NB : *attention au « . » en fin de commande, qui indique le répertoire courant (celui dans lequel se trouve le Dockerfile).*

Docker va alors exécuter chaque étape du Dockerfile, en commençant par le téléchargement de l'image depuis le hub Docker. Patientez pendant la création de l'image. Une fois celle-ci créée, vous pouvez exécuter la commande :

```
> docker images
```

Celle-ci permet de lister les différentes images qui sont disponibles sur votre hôte Docker. Vous devriez retrouver l'image créée un peu plus tôt : Fig.7.

Instanciation de l'image dans un conteneur Docker

Pour instancier un nouveau conteneur à partir de l'image, il suffit d'utiliser la commande suivante :

```
> docker run --name "nom du conteneur" -p 80:5004 aspnet5programmez
```

Le paramètre « -p » permet d'indiquer que le port 80 de la machine hôte doit être mappé sur le port 5004 du conteneur. Le dernier paramètre correspond au nom de l'image à instancier ! Il ne reste plus qu'à ouvrir le port 80 sur la machine hôte (depuis le portail Azure, dans les paramètres de la VM) et vous pourrez alors parcourir votre application ASP.NET 5, qui s'exécute dans un conteneur Docker Linux, sur Microsoft Azure !

Pour stopper le conteneur, il vous suffira d'utiliser la commande :

```
> docker stop "nom du conteneur"
```

Conclusion

Dans cet article, nous avons vu comment utiliser la plateforme Microsoft Azure pour créer un hôte Docker sous Linux et y déployer un conteneur hébergeant une application ASP.NET 5, à partir de l'image mise à disposition par Microsoft dans le hub Docker.

Il s'agit d'un premier aperçu de ce que la plateforme de Cloud Microsoft propose en termes d'utilisation de la technologie Docker et, comme nous l'avons vu en introduction, énormément de nouveautés sont attendues dans les prochains mois, notamment avec l'arrivée des conteneurs Windows Server 2016, pilotables par Docker.

A terme, on peut imaginer des scénarios très intéressants, notamment autour de Docker Swarm, la technologie qui permet d'orchestrer des clusters d'hôtes Docker, avec la possibilité d'avoir, au sein d'un même cluster, des hôtes Docker Linux et Windows, pour des scénarios d'applications hybrides !



```
jcorioland@beedoobuntu: ~/src/AspNet5Programmez/src
jcorioland@beedoobuntu:~/src/AspNet5Programmez/src$ docker images
REPOSITORY                       TAG             IMAGE ID        CREATED            VIRTUAL SIZE
aspnet5programmez                latest          8109a219495d    15 minutes ago     1.047 GB
minecraft_loserboard_aspnet_beta6 latest         57933df80d2a    3 weeks ago        961.5 MB
microsoft/aspnet                 1.0.0-beta6     3c51fef8a8a6    3 weeks ago        730.2 MB
minecraft_loserboard             latest          7d5b4c9f0f7f    3 weeks ago        991 MB
microsoft/aspnet                 1.0.0-beta5     3f72afbf5e5a    5 weeks ago        729.8 MB
jcorioland@beedoobuntu:~/src/AspNet5Programmez/src$
```

Fig.7

REFACTORING TO FUNCTIONAL

Les conférences d'Hadi Hariri, développeur et évangéliste technologique chez JetBrains, sont pour moi des conférences à ne pas manquer à Devoxx France, non seulement du fait de ses qualités indéniables d'orateur, mais aussi de la diversité et de la pertinence des sujets traités.



Jérôme Valloire
SOAT



Cette année, Hadi Hariri a décidé de s'attaquer à la programmation fonctionnelle au travers d'une conférence intitulée "***Refactoring to Functional***" au cours de laquelle il est revenu sur les avantages de ce paradigme de programmation en s'appuyant sur différents exemples concrets.

Pourquoi se tourner vers la programmation fonctionnelle ?

Pour Hadi Hariri, la programmation fonctionnelle doit permettre aux développeurs :

- D'écrire moins de code,
- D'écrire du code plus expressif,
- D'écrire du code plus correct,
- D'écrire du code plus performant.

De plus, et il faut bien se l'avouer, la programmation fonctionnelle est à la mode aujourd'hui, avec l'avènement dans le monde de l'entreprise de langages tels que Scala, Haskell... mais aussi avec l'introduction des lambdas dans Java depuis Java 8. Ainsi, l'approche fonctionnelle, longtemps considérée comme purement académique, connaît aujourd'hui un essor réel, bouleversant complètement nos habitudes de programmation impérative. Par conséquent, sa mise en œuvre requiert un effort certain d'apprentissage pour être efficacement utilisée.

Quel langage utiliser ?

Un langage fonctionnel se caractérise par plusieurs critères :

- L'utilisation de fonctions en tant qu'**objets de première classe** ("*first-class citizens*"), celles-ci étant des données, au même titre que toutes autres données :
  - **Fonctions d'ordre supérieur** ("*High-Order functions*") : qui peuvent prendre une ou plusieurs fonctions comme paramètres et renvoyer une nouvelle fonction comme valeur de retour,
  - **Lambda** : fonctions anonymes nommées ainsi en référence au Lambda-calcul.
- La possibilité de manipuler des **données immuables**.

Il existe de nombreux langages fonctionnels :

- **Lisp**, l'ancêtre de tous les langages fonctionnels, créé dès 1958,
- **Scheme**, un dérivé de Lisp, créé en 1975,
- La famille des langages **ML** (Meta Language), qui a vu le jour à la fin des années 70 et dont les principaux représentants aujourd'hui sont **SML** (Standard ML) et **Caml** (avec **OCaml**),
- **Erlang**, créé en 1989,
- **Haskell**, créé en 1990 et qui a la particularité d'être un langage "fonctionnel pur" (sans effet de bord), ce qui en fait LE langage fonctionnel par excellence aujourd'hui.

En plus de ces langages fonctionnels historiques, on trouve également de nombreux langages multi-paradigmes, qui permettent une approche fonctionnelle :

- **F#**, né en 2002, qui est un langage de programmation fonctionnelle, impératif et orienté objet pour la plateforme .NET,
- **Scala**, apparu en 2003 dans le but de concilier les paradigmes de programmation orientée objet et de programmation fonctionnelle,
- **Clojure,** créé en 2007, qui est un dialecte Lisp pour la plateforme Java,
- **Java 8**, mais également les versions précédentes, avec l'ajout de librairies tierces telles que Guava ou Functional Java.

Et bien d'autres ...

Pour sa présentation, Hadi Hariri a utilisé **Kotlin**, un autre langage basé sur la JVM, qui se rapproche de Scala. A noter que ce langage est développé par JetBrains, d'où son choix "naturel" pour cette conférence.

```
fun basicFunction(parameter: String): String {
    return parameter
}

fun singleLineFunction(x: Int, y: Int) = x + y

fun higherOrderFunction(func: (Int) -> Int) { ... }

val lambda = { (x: Int, y: Int) -> x + y }
```

Pour qu'une fonction soit considérée comme **pure**, les mêmes données d'entrée doivent toujours produire le même résultat en sortie, tout en n'induisant aucun effet de bord. Cette propriété rend possible la **transparence référentielle**, principe selon lequel le résultat d'une opération ne change pas si on remplace une expression par une expression de valeur égale. Un langage fonctionnel est dit "**fonctionnel pur**" s'il est sans effet de bord. Par exemple, dans de tels langages, on ne trouve aucune opération d'affection. C'est le cas du langage Haskell.

L'approche fonctionnelle en action

Une première fonction : itdoessomething ...

Hadi Hariri nous propose tout d'abord la fonction suivante :

```
fun itDoesSomething(elements: List<String>): Map<String, Int> {
  var i = 0
  val results = hashMapOf<String, Int>()
  while (i < elements.size) {
    val element = results.get(elements[i])
    if (element != null) {
      results.set(elements[i], element + 1)
    } else {
      results.set(elements[i], 1)
    }
    i++
  }
  return results
}
```

Comme son nom l'indique, cette fonction fait ... quelque chose !

Mais vous en conviendrez, il n'est pas particulièrement aisé de comprendre le but de cette fonction sans analyser précisément chacune de ses lignes de code, où s'entremêlent données et opérations sur ces données. Cependant, il y a fort à parier que, si vous analysez le code des projets sur lesquels vous travaillez, ce type de code impératif est légion... tout

en espérant que les noms de ces fonctions soient quand même un peu plus explicites ! Une version fonctionnelle de cette fonction est la suivante :

```
fun itDoesSomething(elements: List<String>): List<Pair<String, Int>> {
  return elements.groupBy {
    it
  }.map {
    Pair(it.key, it.value.count())
  }
}
```

Ce code est non seulement plus concis, mais également plus expressif et même sans connaître les fonctions *groupBy* et *map*, on peut commencer à en deviner le but : déterminer le nombre d'occurrences de chaque chaîne de caractères de la liste passée en paramètre.

Un exemple de refactoring étape par étape

Nouvel exemple, avec cette fois-ci une fonction visant à filtrer une liste d'albums de Pink Floyd, le groupe préféré de Hadi Hariri, pour ne conserver que les tops albums, classés 1er aux Royaume-Uni et aux États-Unis.

```
data class Album(val title: String, val year: Int, val chartUK: Int, val chartUS: Int)
val albums = listOf(
  Album("The Dark Side of the Moon", 1973, 2, 1),
  Album("The Wall", 1979, 3, 1),
  Album("Wish You Were Here", 1975, 1, 1),
  Album("Animals", 1977, 2, 3),
  Album("The Piper at the Gates of Dawn", 1967, 6, 131),
  Album("The Final Cut", 1983, 1, 6),
  Album("Meddle", 1971, 3, 70),
  Album("Atom Hearth Mother", 1970, 1, 55),
  Album("Ummagumma", 1969, 5, 74),
  Album("A Sauceful of Secrets", 1968, 9, 0),
  Album("More", 1969, 9, 153))

fun topUSandUK_v1(albums: List<Album>): List<Album> {
  val hits = arrayListOf<Album>()
  for (i: Int in 0..albums.count()-1) {
    if (albums[i].chartUK == 1 && albums[i].chartUS == 1) {
      hits.add(albums[i])
    }
  }
  return hits
}
```

Cette fonction utilise une boucle *for* classique afin de tester une condition sur chacun des albums de la liste. Cependant, on note ici la présence de la variable *'i'*, utilisée comme index de boucle ; il s'agit d'une variable d'état que l'on peut aisément éliminer en utilisant une structure de type *for... in* :

```
fun topUSandUK_v2(albums: List<Album>): List<Album> {
  val hits = arrayListOf<Album>()
  for (album in albums) {
    if (album.chartUK == 1 && album.chartUS == 1) {
      hits.add(album)
    }
  }
  return hits
}
```

En rajoutant un peu de sucre syntaxique, on obtient la fonction suivante :

```
fun topUSandUK_v3(albums: List<Album>): List<Album> {
  val hits = arrayListOf<Album>()
  albums.forEach {
    if (it.chartUK == 1 && it.chartUS == 1) {
      hits.add(it)
    }
  }
  return hits
}
```

Suite à ces modifications, la boucle *for* a été remplacée par une fonction *forEach*, qui reprend le même bloc de code.

On se rapproche du fonctionnel mais un problème demeure, car on continue à se concentrer sur la manière de faire le traitement (le *how*) et non sur ce que l'on veut réellement faire (le *what*).

Il reste donc à éliminer le bloc conditionnel à base de *if* en utilisant une nouvelle fonction, la fonction *filter* avec un prédicat :

```
fun topUSandUK_v4(albums: List<Album>): List<Album> {
  return albums.filter {
    (it.chartUK == 1 && it.chartUS == 1)
  }
}
```

Rien de magique dans ce refactoring : on a simplement extrait du code en termes de traitements et de données, puis mis en place une fonction pour les faire interagir, cette fonction *filter* assurant le parcours de la liste et le test de la condition. Il existe beaucoup d'autres fonctions de ce type qui sont des applications du concept de **Fonctions d'Ordre Supérieur** qui prennent en paramètres d'autres fonctions. Hadi Hariri nous présente également la fonction *map*, qui permet de réaliser simplement des transformations de données :

```
fun topUSandUK_hits_years_v1(albums: List<Album>): List<Int> {
  val hits = albums.filter {
    (it.chartUK == 1 && it.chartUS == 1)
  }
  val years = arrayListOf<Int>()
  hits.forEach {
    years.add(it.year)
  }
  return years
}

fun topUSandUK_hits_years_v2(albums: List<Album>): List<Int> {
  return albums.filter {
    (it.chartUK == 1 && it.chartUS == 1)
  }.map {
    it.year
  }
}
```

Dans cet exemple, on récupère les années correspondantes aux tops albums et au lieu d'utiliser une boucle *for* (le *how*), on utilise la fonction *map* à laquelle on passe la fonction permettant de récupérer la propriété "*year*" de chaque album (le *what*).

Ces refactorings nous permettent également de rendre le code plus concis

et plus expressif, en déléguant une partie de l'écriture de code à ceux qui mettent en place les fonctions de base du langage, les frameworks ou les librairies fonctionnelles. L'approche fonctionnelle permet donc de se concentrer uniquement sur le *what,* rendant le code plus compréhensible.

Et les patterns object-oriented dans tout ça ?

Jusque-là rien de bien nouveau si vous connaissez un peu de Scala ou si vous avez eu l'occasion d'utiliser les Streams de Java 8. Hadi Hariri s'attaque maintenant à certains patterns *Object-Oriented,* pour nous prouver qu'ils peuvent également être modifiés en style fonctionnel, grâce à l'utilisation de fonctions.

Patron de méthode *("template method pattern")*

En approche objet, un patron de méthode définit le squelette d'un algorithme à l'aide de méthodes abstraites dont le comportement concret se trouve dans des sous-classes implémentant ces méthodes.
Voici un exemple d'implémentation traditionnelle de ce pattern :

```
public abstract class Record {
  public abstract fun editRecord()
  public abstract fun persistData()
  public fun checkPermissions() {
    // Check permissions
  }
  public fun edit() {
    checkPermissions()
    editRecord()
      persistData()
    }
  }

  public class CustomerRecord: Record() {
    override fun editRecord() {
      // Edit customer record
    }
    override fun persistData() {
      // Persist customer data
    }
  }

  public class InvoiceRecord: Record() {
    override fun editRecord() {
      // Edit invoice record
    }
    override fun persistData() {
      // Persist invoice data
    }
  }
```

Une telle hiérarchie de classes est-elle réellement nécessaire alors que les seules parties qui diffèrent entre les deux classes d'implémentation sont les fonctions ? En approche fonctionnelle, l'implémentation de ce pattern devient beaucoup plus simple :

```
public class RecordFunctional(val editRecord: () -> Unit, val persistData: () -> Unit ) {
  public fun checkPermissions() {
    // Check permissions
  }
  public fun edit() {
    checkPermissions()
    editRecord()
    persistData()
  }
}
```

Les fonctions sont ici directement passées en paramètres, ce qui permet d'éliminer tout le code "*boilerplate*" de l'implémentation initiale.

Stratégie

Nouvel exemple, avec cette fois le pattern stratégie appliqué à des classes définissant différents algorithmes de tris :

```
public trait SortAlgorithm {
  public fun <T> sort(list: List<T>): List<T>
}

public class QuickSort: SortAlgorithm {
  override fun <T> sort(list: List<T>): List<T> {
    // Processing ...
    return sortedList
  }
}

  public class BubbleSort: SortAlgorithm {
    override fun <T> sort(list: List<T>): List<T> {
      // Processing ...
      return sortedList
    }
  }

  public class Sorter(private val algorithm: SortAlgorithm) {
    public fun <T> sortList(list: List<T>): List<T> {
      return algorithm.sort(list)
    }
  }
```

En approche objet, on définit une interface (*trait* en Kotlin) et différentes classes d'implémentation correspondant aux différents types de tris.
Là encore, on peut utiliser une approche fonctionnelle pour passer directement la fonction, et donc le comportement souhaité, à la méthode de tri du *Sorter* :

```
public class SorterFunctional() {
  public fun <T> sortList(list: List<T>, sortFunction: (List<T>) -> List<T>): List<T> {
    return sortFunction(list)
  }
}
```

La fonction *sortFunction* permet tout simplement de définir l'algorithme de tri, éliminant une fois de plus la nécessité de définir les différentes classes de l'approche objet.

Injection de dépendances

L'utilisation de l'injection de dépendances est monnaie courante aujourd'hui dans nos développements.
Voici un exemple de *Repository* dans lequel on injecte *DataAccess*, une classe assurant la configuration de l'accès à la base de données :

```
public class CustomerRepository(val dataAccess: DataAccess) {
    public fun getById(id: Int): Customer {...}
    public fun getByName(name: String): List<Customer> {...}
    public fun getByEmail(email: String): List<Customer> {...}
}
```

Jusque-là rien d'anormal, les différentes méthodes utilisant *DataAccess* pour réaliser les différentes requêtes.

Cependant, voyons ce qui se passe lorsque le nombre de dépendances augmente, par exemple dans un *Controller* :

```
public class CheckoutController(val cartRepository: CartRepository,
                val shipmentRepository: shipmentRepository,
                val taxService: TaxService) {
    public fun index() {
        // Render Shopping cart with checkout buttons
        cartRepository
    }

    public fun checkout() {
        // Render checkout page including data for payment
        taxService
    }

    public fun shipment() {
        // Render shipment page including data for shipment options
        shipmentRepository
    }
}
```

Ici, chaque méthode utilise une dépendance différente mais il nous faut quand même définir toutes les dépendances. Cela n'est pas vraiment en accord avec les principes SOLID, non?

En fait, nous n'avons pas réellement besoin de ces dépendances, simplement de leurs comportements, ce qu'il est une nouvelle fois possible de refactorer en utilisant une approche fonctionnelle, pour encapsuler ces comportements dans des fonctions :

```
public fun checkoutHandler(checkoutDataFunc: (Int) -> List<CartEntry>, id: Int) {
    val data = checkoutDataFunc(id).take(10)
    // Process data
}
```

Hadi Hariri nous démontre ainsi que l'approche fonctionnelle est parfaitement utilisable dans des scénarios classiques, que nous avons l'habitude de traiter de manière totalement différente dans un style de programmation objet.

Suite dans le N° 190

New-York dans votre salon : une application Maker à la loupe

Si vous avez eu la chance de visiter Big Apple et la fameuse place Times Square, à deux pas de Central Park, vous avez dû être frappé par la taille des écrans géants qui diffusent en permanence des publicités ou des informations. Les plus grandes marques commerciales paient des millions pour apparaître en gros plan sur ces zones publicitaires dynamiques. Ou alors, vous utilisez quotidiennement les transports en commun, et vous êtes attentif aux informations qui défilent à l'intérieur des bus et des rames de métro (prochain arrêt, temps restant jusqu'au terminus, préavis de grève...).



Laurent Vaylet
ingénieur produits chez
National Instruments

Avez-vous déjà réfléchi à la technologie utilisée par ces panneaux ? Ils sont très différents de votre écran de télévision et reposent sur des milliers, voire des millions, de minuscules diodes électroluminescentes (LED) contrôlées par des dizaines de processeurs dédiés. Que diriez-vous de mettre un peu de Times Square dans votre appartement ou votre maison, sans avoir besoin de déménager à New-York ? **Fig.1.**

Aujourd'hui, il est facile de se procurer des matrices LED préassemblées et pré-câblées. Des sites comme Adafruit ou eBay proposent ces matrices en différentes tailles, du 16x32 au 32x64 avec différentes résolutions (le « pitch », ou distance entre deux LEDs voisines). Ces matrices élémentaires peuvent même être chaînées pour obtenir des écrans de grande taille comme ceux de Times Square. ? **Fig.2.**

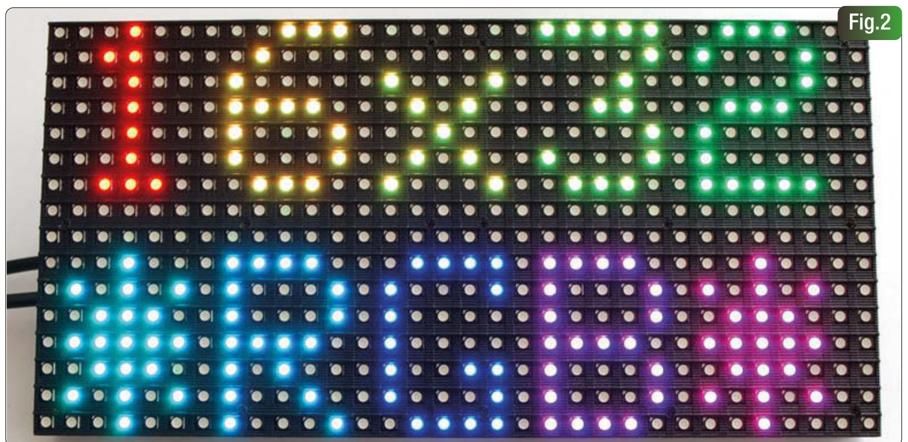
Plus proche de nous, Max Kuznetsov, élève ingénieur en stage chez National Instruments au Royaume-Uni, a utilisé un modèle 32x32 pour concevoir six applications différentes : une galerie d'images, l'affichage de la zone de l'écran sous le curseur de la souris, la diffusion d'une vidéo, un jeu « Snake » pour les nostalgiques du Nokia 3310, le jeu de la vie de Conway et l'affichage du flux vidéo d'une webcam.

Passionné par la culture et le mouvement Maker, Max a mis à la disposition de tous les schémas électroniques et le code utilisé pour parvenir à ces résultats étonnants.

Voyons ensemble comment il a procédé, et les choix technologiques qu'il a effectués pour exploiter au maximum les possibilités de sa matrice. ? **Fig.3.**



Des écrans LED géants sur Times Square à New-York.

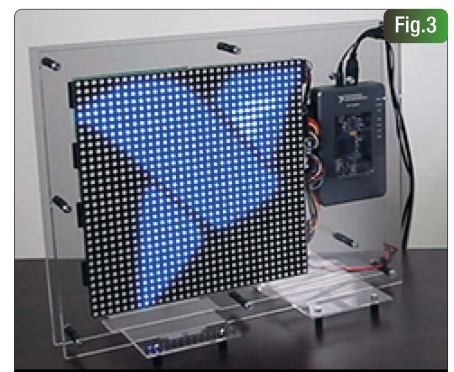


Un exemple de matrice 16x32.

Défis technologiques à relever

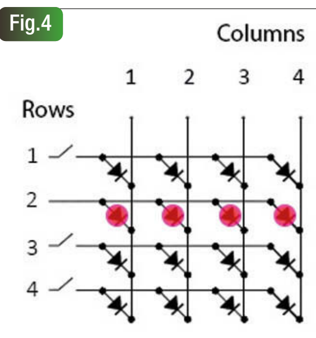
Se procurer les matrices LED sur Internet ou dans des boutiques physiques est facile. Les contrôler est un peu plus compliqué. Il faut choisir les bonnes technologies pour être capable d'utiliser toute la panoplie des possibilités offertes.

La première limite à surmonter concerne le nombre de diodes à piloter. Un simple panneau 16x32 contient 512 LEDs. Il est impensable de

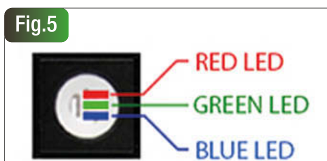


L'application finale, avec le pilotage d'une matrice LED 32x32.

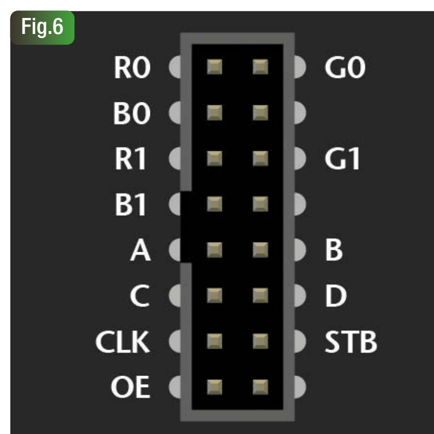
les piloter toutes en même temps, à moins de posséder 512 broches d'entrées/sorties numériques et une alimentation capable de délivrer le courant nécessaire à l'allumage simultané des diodes !
Habituellement, une matrice 16x32 est donc divisée en 8 sections entrelacées : la première section est composée des lignes 1 et 9, la deuxième section des lignes 2 et 10, et ainsi de suite jusqu'à la septième section composée des lignes 7 et 16. Avec cette répartition, chaque section contient 2x32, soit 64 LEDs.
L'astuce utilisée pour minimiser le courant et le nombre de broches nécessaires consiste à rafraîchir les sections les unes après les autres. Si le rafraîchissement est suffisamment rapide, le phénomène de persistance rétinienne fait croire au cerveau que toutes les LEDs sont allumées simultanément. Le choix d'un entrelacement des lignes, à la place de lignes consécutives, améliore le confort visuel en minimisant le clignotement des sections. Fig.4.
Si vous retournez une matrice pour voir ce qui se cache derrière, vous trouverez des puces semblables au très répandu 74HC595, capables de gérer 16 informations simultanément avec un courant constant. Une matrice 16x32 contient typiquement 12 de ces puces pour être capable de piloter 12x16, soit 192 lignes simultanément. Vu autrement, c'est exactement ce qu'il faut pour allumer ou éteindre les 3 composantes rouge/vert/bleu de 64 LEDs multicolores. La boucle est bouclée. Il ne reste plus qu'à trouver un contrôleur suffisamment puissant pour gérer le pilotage de ces 16 puces. Fig.5.
Dans le détail, le contrôleur sélectionne la bande de LEDs à « dessiner » grâce à trois bits A, B et C, qui permettent de représenter les 8 valeurs associées aux 8 bandes disponibles. Une fois l'adresse configurée et la bande sélectionnée, le contrôleur génère 192 bits de données (24 octets) pour piloter les deux lignes de diodes de la section. Il incrémente ensuite l'adressage, génère à nouveau 192 bits de données et répète l'opération jusqu'à atteindre la 8ème section. Il réinitialise alors l'adresse pour rafraîchir l'affichage à partir de la première section.Fig.6.
Cette technique, bien que très simple à mettre en œuvre, présente l'inconvénient de ne pas offrir de pilotage précis de l'intensité des LEDs. Elles sont soit allumées, soit éteintes. Un contrôle de type PWM (Pulse-Width Modulation) permet d'ajuster l'intensité de chaque diode, mais requiert en contrepartie une puissance de calcul largement supérieure à ce que peut offrir un microcontrôleur classique (40 à 50 MHz minimum d'après des essais). Il est alors obligatoire de s'orienter sur des solutions de type FPGA pour atteindre les cadences nécessaires.



Le contrôleur ne pilote pas toutes les lignes simultanément.



Les trois composantes Rouge/Vert/Bleu (RGB).



Le bornier utilisé pour piloter la matrice LED (accessible au dos de la matrice).

Choix du matériel

En prenant en compte toutes ces considérations, un contrôle on/off des LEDs peut être effectué par un microcontrôleur de type Arduino Uno tant que le nombre de diodes reste faible. Dès que le nombre de LEDs augmente, par exemple en chaînant plusieurs matrices les unes derrière les autres pour obtenir une surface d'affichage plus grande, des microcontrôleurs plus puissants deviennent nécessaires. C'est aussi le cas si vous souhaitez pouvoir contrôler finement l'intensité de chaque LED avec un pilotage de type PWM. Pour toutes ces raisons, et vu les contraintes imposées par ses applications, Max Kuznetsov s'est orienté vers NI myRIO pour le choix du contrôleur.

Plus de détails sur le matériel utilisé

La plateforme myRIO repose sur une architecture Xilinx Zynq avec une puce regroupant un processeur ARM Cortex-A9 double-cœur à 667 MHz et un FPGA à 40 MHz. Couplés au système d'exploitation Linux RT et aux 60 entrées/sorties analogiques et numériques disponibles, ces deux cerveaux permettent de piloter aisément des matrices 32x32 (ou n'importe quel système robotique ou mécatronique de manière générale).
Les deux cœurs du processeur ARM gèrent la partie temps réel et le déroulement du programme (gestion des interactions avec l'utilisateur, par exemple). Le FPGA, quant à lui, se spécialise dans le pilotage à très haute vitesse de l'allumage des LED. Ses 40 MHz lui permettent de générer 256 valeurs différentes pour chaque composante rouge/vert/bleu, pour un total de 16,8 millions de couleurs par LED. C'est grâce à cette diversité que les panneaux de Times Square arrivent à produire des images aussi réalistes à partir de « simples » LEDs.
Si vous souhaitez aller plus loin, myRIO ne se limite pas au pilotage de diodes. Le WiFi intégré permet de communiquer avec d'autres appareils (tablettes, smartphones, microcontrôleurs tiers) et participer ainsi à l'Internet des Objets.



Ressources complémentaires et liens utiles

Démonstration en vidéo de l'application : https://www.youtube.com/watch?v=4JmxbJl-Daw
Code source complet et documentation de l'application : https://decibel.ni.com/content/docs/DOC-37231
Documentation des panneaux de LEDs (Adafruit) :
https://learn.adafruit.com/32x16-32x32-rgb-led-matrix?view=all
Ressources d'apprentissage pour myRIO : http://www.ni.com/tutorial/14621/fr/
Communauté myRIO : https://decibel.ni.com/content/community/academic/products_and_projects/myrio
Plus d'informations sur myRIO : http://www.ni.com/myrio/f/
Demandez une unité d'évaluation myRIO : http://www.ni.com/myrio/evaluate/f/

CloudUnit : le PaaS Java Open Source

CloudUnit est une nouvelle plateforme permettant de déployer, exécuter et gérer des applications Java et Java EE. Cette nouvelle plateforme est développée par la startup Treeptik, et est actuellement en bêta privée, mais sera prochainement fournie en open source et disponible sur GitHub. L'article présente la plateforme ainsi qu'un exemple de déploiement d'une application Spring afin d'illustrer certaines fonctionnalités.



Fabien AMICO
Founder & CTO
treeptik.fr

CloudUnit est un PaaS spécialisé dans le déploiement et la gestion d'applications Java et Java EE. C'est une plateforme qui a fait le choix d'être mono langage afin de traiter spécifiquement les problématiques liées au langage Java, et de fournir ainsi un outillage plus adapté. CloudUnit s'intègre dans la mouvance Devops qui, comme le rappelle le dossier du magazine *Programmez* de décembre 2014, est une philosophie globale qu'il faut instrumenter avec les bons outils et les bonnes personnes - ce dernier dossier fait aussi la présentation de BlueMix d'IBM offrant une approche différente des problématiques PaaS.

Les différences de CloudUnit

CloudUnit fournit un panel de fonctionnalités spécialisées pour les applications Java et Java EE ce qui permet de mettre à disposition des outils d'administration simplifiés, avec la possibilité de modifier simplement la version de la JVM ou de passer des options à celle-ci. Les applications, dans CloudUnit, sont constituées d'un serveur d'application (**Tomcat** ou **Jboss**) et de différents modules (**Mysql, PostgreSQL, Redis, MongoDB** ....) avec pour chacun, un manager, qui permet de le configurer. La plateforme permet de déployer les applications Java, soit via l'interface Web, soit en ligne de commande avec le Command Line Interface, soit en poussant le code source avec un "git push" sur une branche distante, et c'est la plateforme qui, à ce moment là, se charge de construire l'application avec Maven et de la déployer. Avec CloudUnit, chacun travaille en fonction de ses préférences. Il est aussi possible de cloner et dupliquer à l'infini une application avec ses modules. CloudUnit permet également de consulter les Logs en ligne ainsi que toute l'activité CPU, Mémoire, Réseau, IO en temps réel. Depuis sa création, CloudUnit repose sur **Docker** - une technologie Open Source partenaire de la startup, qui a permis le développement rapide de la plateforme, et qui offre pour l'avenir une extensibilité importante. En outre, la possibilité, pour ceux qui le souhaitent, d'extraire les applications de CloudUnit et de les basculer dans des environnements de production basés à 100% sur du Docker, comme Mesosphere ou Kubernetes. Coté Docker, CloudUnit n'utilise pas d'outils spécifiques tels que des ordonnanceurs. Ceci s'explique par le fait que la plateforme est elle-même un ordonnanceur de containers spécialisés pour les applications Java et Java EE. L'architecture microservice est à l'honneur en alignement avec les préconisations de Docker. Une application standard dans CloudUnit se construit avec un minimum de 7 containers, cependant cela s'effectue en toute transparence pour le développeur, qui ne voit que son application. Coté stack applicative, toute la plateforme est développée en Java avec notamment ElasticSearch, LogStash et une grosse tuyauterie Spring. Quelques API comme cAdvisor sont utilisées pour le monitoring des containers.

Déploiement d'une application Spring dans CloudUnit

Dans cet article nous allons déployer la fameuse application "Petclinic", exemple développé par Spring qui va nous permettre d'illustrer quelques unes des fonctionnalités de CloudUnit comme : la création d'application, l'ajout d'une base de données, la configuration de la JVM et le déploiement.

En pré-requis, il faut avoir un accès à la plateforme CloudUnit. Cela peut se faire de deux manières soit vous bénéficiez déjà d'un accès sur votre Cloud privé, soit vous souhaitez avoir un accès sur le Cloud public par simple demande sur le site www.cloudunit.fr. Une fois en possession de votre accès, il faut ensuite cloner ou copier le projet qui va nous servir d'application d'exemple; pour cela nous allons nous baser sur l'exemple de ligne de Spring.

https://github.com/spring-projects/spring-petclinic

Une version de Java 8 ainsi qu'une version de Maven sur votre machine vous sera nécessaire.

Création de l'application

Il faut dans un premier temps se connecter à la plateforme avec les identifiants fournis. Une fois connecté, l'interface Web laisse apparaître le Dashboard de CloudUnit qui permet entre autres de lister les applications existantes et d'en créer de nouvelles. Dans CloudUnit, une application est identifiée par un nom et un type de serveur qui peut être pour l'instant une des versions de Tomcat, ou une des version de Jboss. Dans la zone de saisie **App Name** saisir "**petclinic**" comme nom d'application et choisir **Tomcat 8** dans la liste des serveurs. En quelques secondes l'application est créée et un nouveau bloc apparaît dans le dashboard ***Fig.1.***

A ce moment-là, un serveur Tomcat est créé et démarré ainsi qu'un dépot Git qui peut permettre de partager l'application et de déployer seulement des modifications.

Le nouveau bloc fournit des informations sur l'application avec son statut (Start / Stop / Pending), le modèle du serveur d'application, et

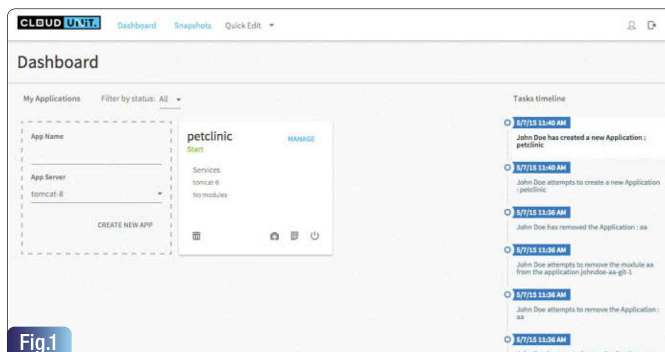


Fig.1

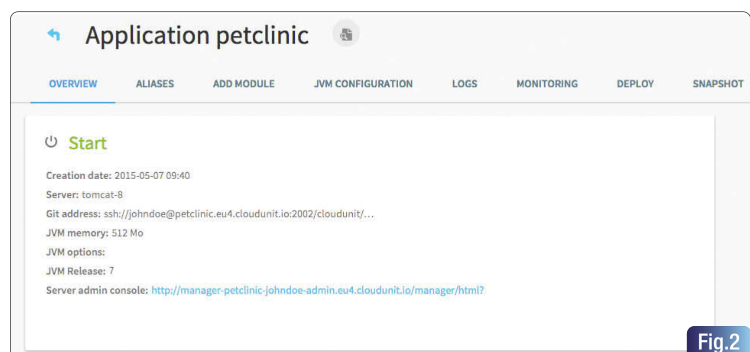


Fig.2

les modules installés. Le lien **MANAGE** permet d'avoir accès à toute la configuration et la gestion de l'application ***Fig.2***. Le premier onglet **OVERVIEW** fournit un maximum de détails sur l'application et les modules qui seront installés par la suite. La barre de menu présente au dessus permet d'avoir accès à toutes les fonctionnalités liées à l'application.

Ajout de module Mysql

Le menu **ADD MODULE** présent dans la barre de menu permet d'ajouter des modules à l'application tels que Mysql, PostgreSQL ou MongoDB, et bien d'autres prochainement ***Fig.3***. Pour l'exemple de l'application PetClinic nous allons installer un module Mysql. L'ajout du module Mysql déclenche l'installation d'une instance complète du SGBD, ainsi que la création d'une première base de données qui porte le même nom que l'application.

Une fois le dispositif installé, un e-mail est envoyé à l'utilisateur avec l'ensemble des informations d'accès à la base de données. Ces informations sont aussi disponibles dans l'interface Web au niveau du premier onglet qui contient le récapitulatif de toutes les informations de l'application. Pour chaque module installé, un outil de gestion individuel est installé automatiquement et accessible depuis le navigateur. Pour le module Mysql, le manager installé est phpMyAdmin qui permet entre autres de configurer, initialiser la base installée, et en créer d'autres en fonction des besoins. Pour accéder à cet outil il faut utiliser les identifiants de la base de données fournis dans le mail ou dans l'interface Web.

Déploiement de l'application

L'application *Spring Petclinic* étant maintenant construite en Java 8 par défaut, il faut ajouter cette configuration dans l'interface d'administration. L'onglet **JVM CONFIGURATION** permet de choisir entre le JDK 7 ou 8 et permet aussi de passer des variables à la JVM, par exemple, pour changer les profils de Spring. Dans notre exemple, il faut choisir la version 8.

A ce niveau d'avancement la plateforme est prête à recevoir l'application. Reste seulement à ajouter dans l'application les informations de la base de données nouvellement créée et à déployer l'application.

Dans le **pom.xml,** il faut décommenter la dépendance MYSQL et commenter la dépendance HSQL. Ensuite il faut mettre les informations de la base de données dans l'application.

Pour cela, il faut éditer le fichier : **src/main/resources/spring/data-access.properties** supprimer les informations du driver HSQL, et configurer le driver MYSQL avec les bonnes informations de connexion à la base de données. Le fichier doit correspondre à l'exemple en ***Fig.4*** avec les bons user/password ainsi que la bonne url JDBC fournie dans le mail ou dans l'interface Web.

Une fois l'application correctement configurée, il ne reste plus qu'à la construire avec Maven pour la déployer avec l'interface WEB. Il est aussi possible de déployer l'application avec un git push sur la branche distante visible dans le premier onglet **overview**. Pour construire l'application avec Maven, il faut lancer une commande **mvn package** à la racine du projet. Cette commande génère l'archive au format war que vous allez ensuite déployer. Le menu **DEPLOY** permet de le faire très simplement, en sélectionnant le fichier .war dans le répertoire target de votre application.

Une fois le fichier sélectionné, il faut cliquer sur le bouton DEPLOY qui lance l'upload du fichier sur le serveur ainsi que le déploiement dans le serveur.

Une fois l'application déployée, elle est instantanément accessible en ligne via un nom DNS configuré par défaut mais qu'il est possible de modifier dans l'onglet **ALIAS**. L'application est accessible depuis la console d'administration Web via une nouvelle icône, juste à coté du nom de l'application. Un clic sur cette icône ouvre l'application dans un nouvel onglet du navigateur.

Conclusion

Cette mise en oeuvre simple nous permet d'illustrer les fonctionnalités de base de CloudUnit et illustre la puissance de ce PaaS, qui nous a permis de mettre en ligne une application Web écrite en Java en quelques minutes - temps de modification de l'application compris !

D'autres fonctionnalités comme les snapshots, une gestion avancée des Logs applicatifs ou du monitoring ***Fig.5***, permettent une amélioration de la productivité dans le travail quotidien des équipes Dev et Ops. Un outil Command Line Interface (CLI) permet, pour ceux qui le préfèrent de travailler, en ligne de commande, mais aussi d'automatiser tous les traitements en exécutant des scripts.

L'application est aujourd'hui disponible gratuitement en Cloud public et en Cloud privé. Pour en savoir plus : http://www.cloudunit.fr.

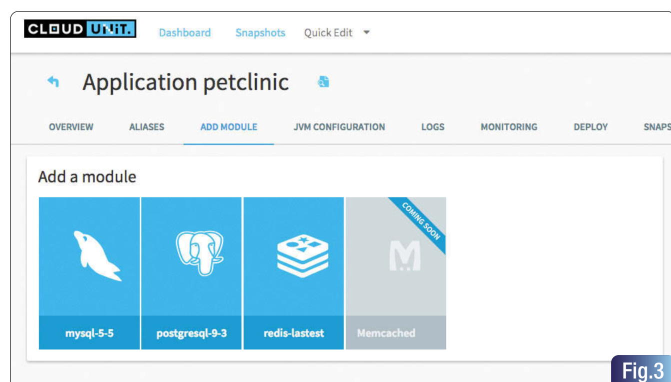


Fig.3



```
#--------------------------------------------------------------------------
# HSQL Settings
jpa.showSql=true

#--------------------------------------------------------------------------
# MySQL Settings

jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://treeptik-petclinic-mysql-5-5-1.mysql-5-5.cloud.unit:3306/petclinic
jdbc.username=adminu28mivba
jdbc.password=kg4ev8bn

# Properties that control the population of schema and data for a new data source
jdbc.initLocation=classpath:db/mysql/initDB.sql
jdbc.dataLocation=classpath:db/mysql/populateDB.sql

# Property that determines which Hibernate dialect to use
# (only applied with "applicationContext-hibernate.xml")
hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Property that determines which database to use with an AbstractJpaVendorAdapter
jpa.database=MYSQL
~
~
```

Fig.4

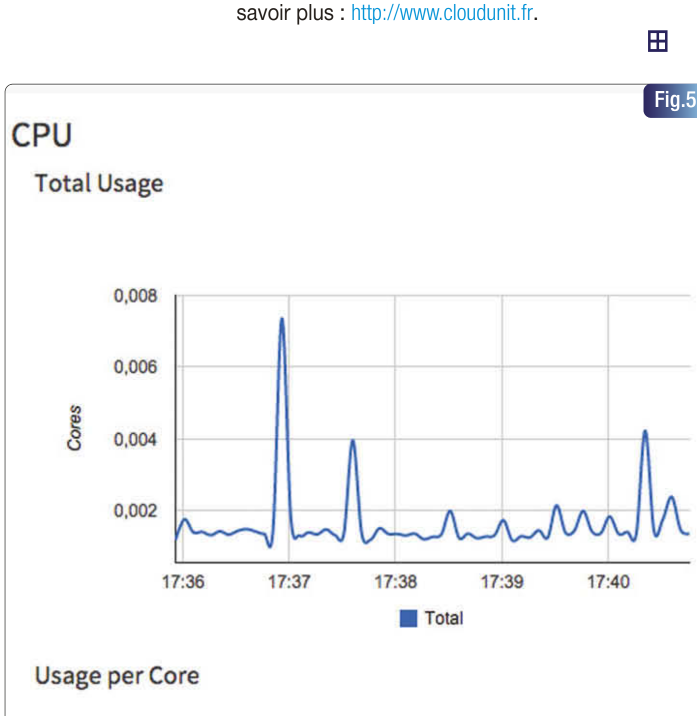


Fig.5

Le Deep Learning pas à pas : les concepts (1/2)

Level 400



Pirmin Lemberger
Coécrit avec Manuel Alves
SQLi


(c) Artomatix

Des labos de R&D à la vie quotidienne

L'image vous rappelle bien quelque chose ? On dirait..., mais oui, c'est la Nuit étoilée de Van Gogh ? Une Nuit étoilée où le Golden Gate Bridge remplace cependant le village bucolique de Saint Remy-de-Provence. Un simple pastiche « à la manière de » qui n'a a priori rien d'extraordinaire, si ce n'est que cette image a été construite numériquement à partir d'une simple photo du célèbre pont de San Francisco et d'une reproduction du chef d'œuvre impressionniste. Ce tour de passe-passe a été réalisé par Artomatix, une start-up irlandaise fondée en 2014 et spécialisée dans la conception de graphismes réalistes pour le cinéma et les jeux vidéo.

En 2012, Google a surpris la communauté du Machine Learning en démontrant que son Google Brain était capable de découvrir, par lui-même, des concepts de haut niveau tels que des visages, des corps humains ou des images de chats, ceci en épluchant des millions d'images glanées sur YouTube. Ce résultat est remarquable, car jusque-là les techniques de reconnaissance d'images se basaient sur des approches dites supervisées, chaque image devant être explicitement désignée comme contenant un visage humain, une tête de chat, etc. Le tour de force des équipes de Google a été de court-circuiter cette étape de labélisation manuelle (tagging). L'enjeu pour Google est énorme, car il s'agit ni plus ni moins que de faire passer à l'échelle ses algorithmes d'apprentissage en tirant parti de la manne d'images disponibles sur le Web qui, dans leur immense majorité, ne sont évidemment pas taguées.

Le point commun entre ces deux applications du Machine Learning ? Les deux utilisent une classe d'algorithmes d'apprentissage automatique que l'on appelle le **Deep Learning**.

Sorti des labos de R&D depuis quelques années, le Deep Learning investit progressivement notre quotidien : la reconnaissance vocale de l'assistant Siri d'Apple, le tagging automatique de morceaux de musique, la synthèse vocale avancée, le légendage automatique d'images et même la conception de nouvelles molécules pharmaceutiques, toutes ces applications mettent aujourd'hui en œuvre des techniques de Deep Learning.

Cet article comprend deux parties. Dans la première partie, nous présenterons en guise d'illustration l'une de ces architectures profondes : les Deep Belief Networks. Notre objectif est d'en donner une présentation conceptuelle détaillée et d'expliquer comment certaines avancées récentes sont parvenues à surmonter d'anciennes difficultés inhérentes aux systèmes de neurones. Une deuxième partie abordera les problèmes liés à l'implémentation des architectures profondes en général.

Cet article présuppose une connaissance rudimentaire des concepts du Machine Learning.

Flash-back

L'idée de construire des **réseaux de neurones** (RN) artificiels n'est pas neuve, elle remonte à la fin des années 50. Très schématiquement l'idée consiste à s'inspirer du fonctionnement du cortex visuel des animaux. Dans une version élémentaire, chaque neurone i d'un tel réseau possède un niveau d'activation x_i compris entre 0 et 1. Le schéma d'interconnexion entre neurones définit l'architecture du réseau. Une architecture classique consiste à organiser les neurones en couches successives avec des interconnexions limitées aux couches adjacentes comme le montre la **Fig.1** (a).

Dans le cadre d'un **apprentissage supervisé**, un exemple classique d'utilisation d'un RN est celui d'un système chargé de classer des images de chiffres manuscrits. Dans cet exemple les niveaux d'activations $x_i^{(1)}$ des neurones $i = 1,..., k$ de la **couche d'entrée** correspondent aux niveaux de gris des pixels de l'image, k étant le nombre de pixels des images. La **couche de sortie** est en l'occurrence constituée de neurones y_j, $j = 0,1,..., 9$ qui correspondent aux dix chiffres que l'on peut attribuer à chaque image d'un ensemble d'entraînement. Les niveaux d'activation des neurones sont déterminés récursivement, couche par couche. Ceux de la couche $n + 1$ sont calculés à l'aide d'une **fonction d'activation** φ en fonction des niveaux d'activation des neurones de la couche n pondérés par certains **poids synaptiques** w_{ij} :

$$x_j^{(n+1)} = \varphi\left(\sum_{i=1}^{k} w_{ij}\, x_i^{(n)}\right) \qquad (1)$$

La somme porte sur tous les neurones i de la couche n connectés au neurone j de la couche $n + 1$, voir la **Fig.1** (b). La fonction d'activation φ est typiquement une fonction telle que la sigmoïde $x \longrightarrow \varphi(x)$ $sigm\ (x)$ représentée dans la **Fig.2**. Elle est croissante, différentiable, non-linéaire et prend ses valeurs entre 0 et 1.

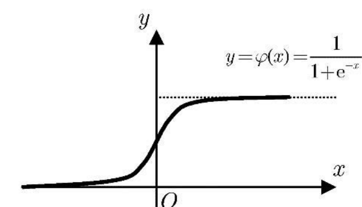
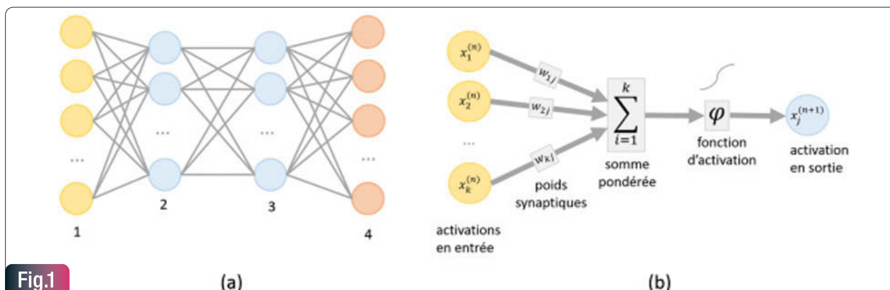


Figure 2 : fonction d'activation sigmoïde

L'**entraînement** du réseau consiste à trouver des poids synaptiques w_{ij} tels que la **couche de sortie** permette de classer avec précision les images d'un ensemble d'entraînement. On espère naturellement que le RN présentera des capacités de **généralisation** sur des exemples qu'il n'a jamais rencontrés.

Une question se pose d'emblée : « De tels poids w_{ij} existent-ils toujours, quel que soit l'objectif assigné au RN ? » Par chance, un



(a) un RN organisé en 4 couches (b) le mécanisme d'activation d'un neurone

résultat mathématique connu sous le nom de théorème d'approximation universel, garantit que la chose est effectivement possible, même pour un réseau ne comportant qu'une seule couche cachée, à condition toutefois que φ soit non-linéaire et qu'un nombre suffisant de neurones soient mis en jeu en fonction de la marge d'erreur tolérée. En revanche si l'on se limite à des fonctions d'activation φ linéaires, le réseau fonctionnera comme une régression linéaire ordinaire et ne sera par conséquent d'aucune utilité pour prédire des phénomènes non linéaires complexes.
Reste à construire un algorithme qui fournit une bonne approximation des poids w_{ij} en un temps acceptable.
Jusqu'à il y a peu l'algorithme phare pour l'entraînement des systèmes de neurones multicouches était l'algorithme dit de **rétropropagation**. Pour fixer les idées, restons sur l'exemple des digits. Pour une image $\mathbf{x} = (x_1,..., x_k)$ en entrée et un ensemble de poids synaptique **w** on commence par définir une **fonction de coût** *C* qui mesure l'écart entre les prédictions $y_j(\mathbf{x})$ des neurones de sortie et les **valeurs cibles** t_j spécifiées dans l'ensemble d'entraînement :

$$C(\mathbf{x}\,;\mathbf{w}) = \sum_{j=0}^{9} \left(y_j(\mathbf{x}) - t_j\right)^2 \qquad (2)$$

Naturellement $y_j(\mathbf{x})$ est une fonction extrêmement compliquée de la configuration **x** en entrée et des poids **w**, spécifiée par l'itération de (1). Ce que l'on cherche à minimiser en principe c'est la somme ou la moyenne $C_{\mathrm{MSE}}(\mathbf{w})$ de ces erreurs sur toutes les configurations **x** de l'ensemble d'entraînement E :

$$C_{\mathrm{MSE}}(\mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{E}} C(\mathbf{x};\mathbf{w}) \qquad (3)$$

Une **descente de gradient** consiste à calculer la direction dans l'espace des poids **w** dans laquelle la décroissance de $C_{\mathrm{MSE}}(\mathbf{w})$ est maximale. Cette direction est naturellement donnée par l'opposé du gradient ? $C_{\mathrm{MSE}}(\mathbf{w})$. Si tout se passe bien, c.-à-d. qu'il n'y a pas de minima locaux, on s'approchera du minimum de $C_{\mathrm{MSE}}(\mathbf{w})$ par itérations successives de petites corrections apportées à **w** :

$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} - \alpha\nabla C_{\mathrm{MSE}}\left(\mathbf{w}^{(\mathrm{old})}\right) \qquad (4)$$

Le coefficient permet d'ajuster la vitesse d'apprentissage, le cas échéant dynamiquement. Dans la pratique cependant, le calcul de la somme sur toutes les configurations $\mathbf{x} \in$ E est impossible, car beaucoup trop coûteux en temps de calcul. Pour remédier à cette situation, on utilise une descente de gradient stochastique (SGD) qui revient à approximer à chaque étape le gradient ? $C_{\mathrm{MSE}}(\mathbf{w})$ par le gradient ? *C* (**x**;**w**) d'un seul échantillon **x** tiré au hasard dans l'ensemble d'entraînement [1] E. Cette stratégie qui peut paraître audacieuse a priori fonctionne, car, intuitivement, les erreurs occasionnées par cette approximation se compensent sur le long terme. L'algorithme de SGD a fait ses preuves dans de nombreux problèmes d'optimisation, et des résultats théoriques viennent étayer cette intuition dans certains cas particuliers.
Reste à calculer toutes les composantes $\partial C(\mathbf{x};\mathbf{w})/(\partial w_{ij})$ du gradient ? *C* (**x**;**w**). La calcul est simple dans son principe puisqu'il ne s'agit ni plus ni moins que de calculer la dérivée d'une fonction composée un peu compliquée définie récursivement par (1) et (2). C'est là qu'intervient l'algorithme de **rétropropagation**, celui-ci permet d'organiser efficacement ce calcul de dérivée. Il s'avère que les dérivées par rapport aux w_{ij} associés à la couche de sortie sont élémentaires. Le reste du calcul procède par induction, car on montre en effet que les dérivées par rapport aux w_{ij} de la couche $n - 1$ sont calculables dès lors celles de la couche *n* ont déjà été calculées. Ce calcul à reculons est à l'origine du nom de l'algorithme.
Tout astucieux qu'il soit l'algorithme de rétropropagation souffre cependant de deux inconvénients majeurs :

- L'expérience montre que le temps d'entraînement d'un RN croît rapidement lorsque le nombre de couches augmente. C'est d'ailleurs l'une des raisons pour lesquelles à partir des années 1990 les RN ont été remisés au profit d'autres algorithmes non linéaires moins gourmand en ressources comme les SVM.
- Les RN n'échappent pas au problème central du Machine Learning : le **surapprentissage**.

Ces résultats empiriques sont en partie corroborés par des résultats en théorie de la complexité qui démontrent que l'entraînement d'un RN est un problème complexe dans un sens précis[2].
Pour progresser, de nouvelles idées sont nécessaires. Après plusieurs décennies de stagnation, c'est G. E. Hinton[3] et son équipe qui, en 2006, ont fait la principale percée dans ce domaine.

Les idées nouvelles de Hinton

L'article fondateur de Hinton contient une série d'idées et de stratégies innovantes. Ce paragraphe les présente sur un plan intuitif, le suivant rentre dans les détails.

Un modèle génératif

Dans un contexte de **classification supervisée**, où l'on essaie d'apprendre une relation entre des observations **x** en entrée (les images de chiffres) et des labels *y* en sortie (les chiffres '0' à '9'), la stratégie la plus courante consiste à construire un **modèle discriminant**, c.-à-d. un modèle pour les probabilités conditionnelles $p(y|\mathbf{x})$ d'observer un label *y* lorsqu'on connaît l'entrée **x**. La régression logistique rentre par exemple dans ce cadre. Par contraste, la stratégie de Hinton et al. consiste à élaborer un modèle génératif, c.-à-d. un modèle pour la distribution de probabilité conjointe $p(y, \mathbf{x})$ des images **x** et des labels *y*. Autrement dit, on cherche à construire un RN capable d'apprendre puis de générer simultanément les images **x** et les labels *y* avec une distribution de probabilité proche de celle observée dans un ensemble d'entraînement.
Une fois le RN entraîné on pourra l'utiliser d'une part pour reconnaître des images, c.-à-d. associer un chiffre *y* à une image **x** (on cherche le y qui maximise $p[y|\mathbf{x}]$) ou, inversement, pour générer des images **x** correspondant à un chiffre *y* (on échantillonne $p[\boldsymbol{x}|y]$). À ce stade, nous invitons vivement le lecteur à jouer avec l'application mise en ligne par Hinton qui illustre ces deux processus de manière visuelle et dynamique. Comme le veut l'adage : une image vaut mille mots, alors une vidéo...

Une initialisation rapide du RN puis un fine-tuning lent

L'architecture de RN utilisée pour réaliser le modèle génératif de Hinton et coll. s'appelle un **Deep Belief Network** (DBN). Elle contient deux parties représentées dans la **Fig.3**.
La première partie du DBN est constituée d'un ensemble de couches de compression qui convertissent les données entrées **x** en une représentation abstraite $\mathbf{x}_{\mathrm{abstrait}}$. La seconde partie convertit cette représentation en labels de classification *y*.
La première partie a pour objectif d'apprendre la distribution des données **x** présentées en entrée sans tenir compte des labels *y*. Elle est constituée d'une succession de couches dont chacune contiendra une représentation plus abstraite (ou compressée) que la précédente. Pour ancrer l'intuition considérons un RN chargé de classifier des images. La première couche stockera les niveaux de gris de l'image (l'équivalent de la rétine), la seconde contiendra, par exemple, un encodage des lignes ou des zones de contraste de l'image, la troisième détectera l'existence de certaines formes géométriques simples comme des cercles, la quatrième identifiera certains agencements particuliers de ces figures comme celles qui représentent un '8' formé de deux cercles juxtaposés et

ainsi de suite. Ainsi on automatise en quelque sorte le processus de **feature engineering** manuel du machine learning. Hinton et coll. ont par ailleurs découvert un algorithme rapide pour entraîner cette première section en procédant couche par couche (nous y reviendrons). Une fois entraînée, cette première section du RN contiendra une **représentation hiérarchique** des données en entrée, la dernière couche encodant la représentation $\mathbf{x}_{abstrait}$ la plus abstraite, et aussi, c'est l'idée, c'est la couche la plus utile. Cette première phase peut être conçue comme une **initialisation** efficace du DBN, on l'appelle aussi **pré-entraînement**.
Le rôle de la seconde partie du DBN est de convertir la représentation abstraite et obscure $\mathbf{x}_{abstrait}$ en labels *y* utilisables par exemple dans le cadre d'un apprentissage supervisé. Dans l'exemple des digits cette représentation sera constituée d'une couche de sortie de dix neurones, un neurone par digit. Cette conversion peut être réalisée au moyen d'une **couche logistique** entraînée par une SGD classique. L'entraînement du DBN est considéré comme achevé lorsque la performance du DBN évaluée sur un ensemble de validation distinct de l'ensemble d'entraînement ne progresse plus significativement. Cette seconde étape est appelée le **fine-tuning**, elle est généralement beaucoup plus lente que l'initialisation.

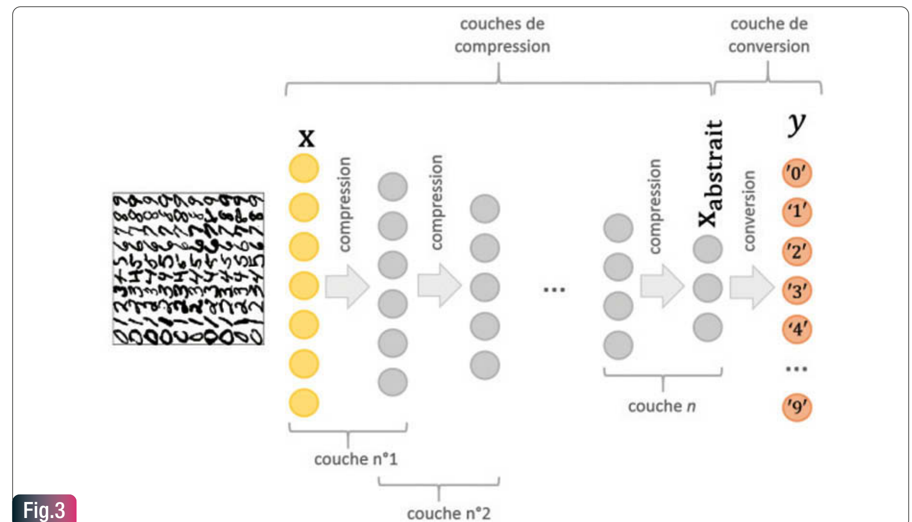


Figure 3 : Un Deep Belief Network qui identifie des images de digits

Les RBM comme briques de base

Pour mener à bien le programme esquissé dans les paragraphes précédents il faut disposer d'une brique de compression (c.-à-d. un morceau de RN) qui implémente les couches de la première section. La principale caractéristique attendue de cette brique est sa capacité à apprendre rapidement une distribution de probabilité spécifiée empiriquement par un jeu d'exemples. Il existe pour cela différentes alternatives[4] qui permettent peu ou prou d'implémenter la même philosophie, mais nous nous restreindrons dans cet article à la description des **Restricted Boltzman Machines** (RBM) utilisées par Hinton et coll. dans leur travail pionnier de 2006.
Le paragraphe suivant explique ce que sont les **RBM** et comment l'algorithme dit de **Contrastive Divergence** de Hinton et coll. exploite astucieusement leurs propriétés.

Zoom sur les RBM

Deux propriétés qui simplifient la tâche

Les RBM sont des RN **stochastiques**, qui peuvent être entraînés sur un mode non supervisé pour générer des échantillons selon une distribution de probabilité complexe spécifiée par des exemples (l'ensemble d'images de digits p.ex.).
Alors que les neurones des RN évoqués dans la section Flash-back ont des niveaux d'activation x_i déterministes compris entre 0 et 1, les neurones des RBM sont des variables aléatoires binaires. Elles sont réparties sur deux couches. L'une de ces couches est appelée la **couche visible**, c'est elle qui encodera les exemples de l'ensemble d'entraînement E. Notons collectivement $\mathbf{v} = (v_1,..., v_k)$ les niveaux d'activation de ces *k* neurones. L'autre est appelée couche cachée, c'est elle qui stockera une forme compressée ou abstraite des données de la couche visible. Notons $\mathbf{h} = (h_1,..., h_l)$ les niveaux d'activation correspondants **Fig.4**.
Par définition, une RBM est définie par la distribution de **probabilité conjointe** suivante[5] sur **v** et **h** :

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \quad \text{où} \quad E(\mathbf{v},\mathbf{h}) = \sum_{i=1}^{k} a_i v_i + \sum_{j=1}^{l} b_j h_j + \sum_{i=1}^{k}\sum_{j=1}^{l} w_{ij} v_i h_j \qquad (5)$$

Les grandes valeurs de *E* (**v**, **h**) correspondent à des probabilités p (**v**, **h**) faibles. La physique statistique utilise des formules similaires pour attribuer des probabilités *p*(**x**) à certaines configurations d'un système thermique. Dans un tel cadre *E*(**x**) s'interprète essentiellement comme l'énergie de **x**. L'intuition derrière la formule (5) est donc que les configurations (**v**, **h**) les plus probables sont celles dont l'**énergie** *E* (**v**, **h**) est minimale. Remarquons que les poids w_{ij} ne connectent que les h_i et les v_j mais pas les h_i entre eux ni les v_j entre eux, c'est là l'origine de l'adjectif « restricted », voir la figure 4. Les paramètres **a**, **b** et **w** sont à optimiser durant l'entraînement. Pour simplifier l'écriture nous les désignerons globalement par .
Ce qui nous intéresse au premier chef c'est la distribution marginale $p_\theta(\mathbf{v}) = \sum_h p(\mathbf{v}, \mathbf{h})$ des

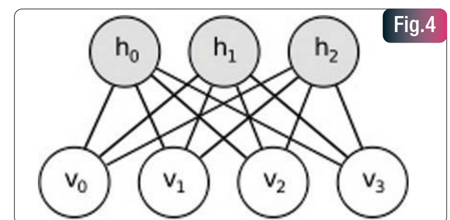


Figure 4 : Une RBM est constituée d'une couche de neurones visibles et d'une couche de neurones cachés

variables visibles **v** de la RBM (la somme porte sur toutes les configurations **h** possibles des neurones cachés et l'indice θ rappelle la dépendance de *p*). L'optimisation de θ a pour objectif de maximiser la ressemblance entre la distribution marginale $p_\theta(\mathbf{v})$ générée par la RBM et la distribution expérimentale observée au sein de l'ensemble d'entraînement E. Pour cela on va maximiser la **vraisemblance** $\prod_{\mathbf{v}\in E} p_\theta(\mathbf{v})$ des valeurs observées **v**. Pour reformuler le problème comme la recherche du minimum d'une fonction de coût qui est une somme de contributions associées à chaque $\mathbf{v} \in$ E on passe au logarithme et on change le signe :

$$C_{\mathrm{NLL}}(\theta) = -\sum_{\mathbf{v}\in\mathcal{E}} \log p_\theta(\mathbf{v}) \qquad (6)$$

L'acronyme NLL signifie Negative Log Likelihood. On peut alors appliquer à C_{NLL} la même stratégie d'optimisation par SGD qu'avec C_{MSE}. Il suffit pour cela de savoir calculer la dérivée de chaque terme dans (6). Pour simplifier les expressions qui suivent on introduit généralement, en parallèle avec l'énergie *E* (**v**, **h**), la notion d'**énergie effective** *F* (**v**) définie par $p_\theta(\mathbf{v}) \equiv e^{-F(\mathbf{v})}/Z$. En utilisant les définitions on trouve :

$$-\frac{\partial}{\partial\theta}\log p_\theta(\mathbf{v}) = \frac{\partial F(\mathbf{v})}{\partial\theta} - \sum_{\mathbf{u}\in\mathcal{E}} p_\theta(\mathbf{u}) \frac{\partial F(\mathbf{u})}{\partial\theta} \qquad (7)$$

Deux propriétés des RBM liées à la forme spécifique de l'énergie *E* (**v**, **h**) définie en (5) vont

grandement nous simplifier la vie. Ce sont elles qui motivent en définitive le choix des RBM comme briques élémentaires des DBN.

- L'**énergie effective** $F(\mathbf{v})$ qui intervient dans (7) **est calculable facilement** au moyen d'une formule explicite :

$$F(\mathbf{v}) = \sum_{i=1}^{k} a_i v_i - \sum_{j=1}^{l} \log\left[1 + \exp\left(-b_j - \sum_{i=1}^{k} w_{ij} v_i\right)\right] \quad (8)$$

Sans le secours de la formule (8) il faudrait calculer une énorme somme sur toutes les configurations **h**, une somme dont le nombre de termes croit exponentiellement avec le nombre l de neurones cachés. Rappelons en effet que $e^{-F(\mathbf{v})} = \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$.

- Les **neurones d'une couche sont conditionnellement indépendants lorsque les activités des neurones de l'autre couche sont fixés**, explicitement (par économie on supprime l'indice θ de p_θ) :

$$p(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{k} p(v_i|\mathbf{h}) \quad (9)$$

$$p(\mathbf{h}|\mathbf{v}) = \prod_{j=1}^{l} p(h_j|\mathbf{v}) \quad (9')$$

Un calcul simple montre par ailleurs que les probabilités conditionnelles individuelles sont données par la fonction sigmoïde :

$$p(v_i = 1|\mathbf{h}) = \text{sigm}\left(-a_i - \sum_{j=1}^{l} w_{ji} h_j\right) \quad (10)$$

$$p(h_j = 1|\mathbf{v}) = \text{sigm}\left(-b_j - \sum_{i=1}^{k} w_{ji} v_i\right) \quad (10')$$

On retrouve donc la fonction d'activation usuelle mais avec une interprétation stochastique. Comme on le verra les propriétés (9,9') et (10,10') sont cruciales pour échantillonner efficacement la distribution p qui intervient dans la somme dans (7).

Le calcul de la somme dans (7) s'avère cependant impraticable pour des grands ensembles d'entraînement[6], car cela prendrait trop de temps. Il faudra par conséquent nous contenter de l'approximer en n'utilisant qu'un nombre limité de termes échantillonnés selon une distribution proche de $p_\theta(\mathbf{v})$. C'est là que les propriétés d'indépendance conditionnelle précédentes viennent à la rescousse.

L'algorithme de Contrastive Divergence

Pour échantillonner une distribution de probabilité multivariée $p(\mathbf{x}) = p(x_1,..., x_r)$ les méthodes dites **Monte-Carlo Markov-Chain** (MCMC) sont souvent indiquées. Elles permettent de générer itérativement une succession infinie d'échantillons $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$... qui, à la longue, seront distribués selon une distribution $p(\mathbf{x})$ souhaitée. Pour les RBM on utilise une version particulière de MCMC dite avec **échantillonnage de Gibbs**.

Concrètement cela signifie que l'échantillon $\mathbf{x}^{(n+1)}$ de la MCMC est généré à partir de l'échantillon $\mathbf{x}^{(n)}$ en r étapes (où r est le nombre de variables que contient **x**). À chaque étape on génère une nouvelle composante x_i en utilisant la probabilité conditionnelle $p(x_i\ \mathbf{x}_{-i})$ où $\mathbf{x}_{-i}$ désigne **x** dont on a retiré la variable x_i. Dans le contexte des RBM qui nous intéresse $\mathbf{x} = (\mathbf{h}, \mathbf{v})$ et les probabilités conditionnelles sont données par les expressions (10) et (10'). Tout l'intérêt de cette propriété d'indépendance conditionnelle propre aux RBM est qu'elle permet de **paralléliser** l'échantillonnage de Gibbs. On va ainsi échantillonner simultanément tous les h_i connaissant les v_j puis, dans un second temps, simultanément tous les v_j connaissant les h_j, explicitement :

$$h_j^{(n+1)} \sim p\left(h_j|\mathbf{v}^{(n)}\right) \quad (11)$$

$$v_i^{(n+1)} \sim p\left(v_i|\mathbf{h}^{(n+1)}\right) \quad (11')$$

Bonne nouvelle, au lieu d'avoir $r = k + l$ étapes, il n'y en a donc finalement que deux étapes parallélisées !

Hélas, en dépit de cette bonne nouvelle du parallélisme, il reste un problème de taille. Notre objectif, rappelons-le, est d'optimiser par itérations successives les paramètres θ de la RBM au moyen d'une SGD. Or chaque itération exige le calcul du gradient (7) qui à son tour demande, du moins en principe, d'attendre que la convergence de l'échantillonnage MCMC vers p_θ s'installe, ce qui est évidemment irréaliste. Hinton et coll. utilisent **deux astuces** pour surmonter cette difficulté. Elles constituent l'algorithme de **Contrastive Divergence**[7] (CD).

Comme la distribution p_θ est sensée être proche de la distribution expérimentale dans E on peut accélérer la convergence de la MCMC en l'initialisant avec un échantillon tiré de E. Ensuite, plutôt que d'attendre patiemment la convergence de la MCMC, on arrête les itérations après un petit nombre d'étapes. Et même, en étant un peu audacieux, après une seule étape ! Ce faisant, on commet évidement une erreur puisqu'on ne génère pas exactement p_θ. L'expérience montre cependant que les itérations de la SDG finissent par lisser cette erreur et que cette approximation grossière reste suffisante pour faire converger la fonction de coût $C_{NLL}(\theta)$ vers un minimum approximatif utile.

Empilement et fine-tuning

Nous voici donc équipé pour entraîner efficacement une RBM. Un DBN consiste comme nous l'avons expliqué en un empilement de RBM et une couche logistique supplémentaire qui convertit le contenu de la couche cachée du dernier RBM en labels de classification.

L'entraînement d'un DBN complet procède alors selon les étapes suivantes :

- La RBM n°1 est entraînée par CD avec les échantillons **v** de l'ensemble d'entraînement E présentés sur sa couche visible,
- Une fois entraînée, cette première RBM fera office de convertisseur/compresseur. À chaque échantillon **v** de l'ensemble d'entraînement E elle associe une configuration **h** de neurones de la couche cachée. On construit **h** en échantillonnant les distributions $p_\theta(h_i\ \mathbf{v})$ avec les paramètres θ appris à l'étape précédente. On peut montrer[8] que les configurations **h** ainsi obtenues sont des versions compressées des **v**. Les **h** sont utilisées comme échantillons d'entraînement présentés à la couche visible de la RBM n°2,
- On itère 1. et 2. sur toutes les RBM du DBN. Une fois toutes les RBM entraînées, la **phase d'initialisation** est terminée. On dispose alors d'un mécanisme complexe de conversion des **v** en une représentation abstraite/compressée sur la couche cachée de la dernière RBM. C'est une forme de **feature engineering automatisé**,
- On peut alors utiliser cette représentation abstraite en entrée d'une couche logistique que l'on entraînera de manière classique avec une SGD supervisée basée sur une fonction de coût C_{NLL} de type maximum de vraisemblance. C'est l'étape dite de **fine-tuning**.

Ce paragraphe clôt la partie conceptuelle de cet article. Cependant le phénomène récent du Deep Learning n'est pas uniquement le résultat de progrès conceptuels. Comme nous le verrons dans la suite, de nombreuses avancées technologiques ainsi que la mise à disposition de nouveaux outils de programmation, promus par les grands acteurs de l'IT, alimentent également cet engouement. ⊞

[1] Il existe des variantes où l'on calcule le gradient de petites sommes que l'on appelle mini-batch.

[2] Il a été démontré que la recherche de poids optimaux dans un RN multicouches est un problème NP-complet.

[3] G.E. Hinton est professeur émérite à l'université de Toronto et chercheur chez Google.

[4] Les principaux sont les auto-encodeurs et les réseaux de convolution, voir cet article p.ex.

[5] La constante Z garantit que p (v, h) est bien une distribution de probabilité, on l'appelle la fonction de partition.

[6] Pour le problème de reconnaissance des digits |E|=50'000 à titre d'exemple.

[7] L'origine de cette terminologie de divergence de Kullback Leibler qui mesure le degré de dissimilarité entre deux distributions de probabilité.

[8] Voir par exemple cet article pour une justification intuitive basée sur l'idée de groupe de renormalisation.

MEAN.IO (1/3)

Ce premier article d'une série consacrée à MEAN.IO va nous permettre d'introduire les principaux concepts du framework, et d'obtenir un environnement de développement opérationnel pour les exercices pratiques des prochains articles.



Luc CLAUSTRES
Consultant Indépendant
Créateur d'Applications Numériques
http://www.digital-innovation.fr

MEAN.IO est un framework Javascript "full-stack" permettant de créer rapidement une application Web de type single-page application (SPA) avec MongoDB, Node.js, Express, et AngularJS (MEAN). Le terme "full-stack" indique qu'il permet de gérer l'intégralité des couches de l'application en Javascript, depuis la base de données, en passant par l'API back-end jusqu'au front-end. Nous verrons que MEAN.IO est un framework au sens "cadre de travail" pour développer une application plutôt qu'au sens ensemble de librairies/composants.
Bien que cet article aborde l'utilisation des technologies MEAN (et d'autres comme Mongoose ou encore Bootstrap) au travers de leur utilisation dans MEAN.IO, je vous conseille en prérequis d'acquérir les bases de programmation dans ces différents frameworks/librairies.

INSTALLATION ET MISE À JOUR

Prérequis

Afin de me simplifier la vie, j'ai l'habitude d'utiliser des solutions pré-packagées pour l'infrastructure de développement ou de production. Pour MEAN.IO j'utilise celles de Bitnami disponibles en environnement Windows, Linux Ubuntu ou de type Cloud comme Amazon. Elles ont l'avantage d'intégrer également Apache, un outil d'administration pour MongoDB nommé RockMongo et Git dont vous aurez besoin pour MEAN.IO. Actuellement je travaille avec Node.js v0.12.4, MongoDB v3.0.3, MEAN.IO v0.5.
Certaines dépendances de MEAN.IO utilisent node-gyp (comme les drivers MongoDB) il faut donc disposer en pré-requis de Python version 2.7.x et d'un compilateur C++ comme GCC sous Linux ou Microsoft Visual Studio C++ sous Windows. L'outil utilisé par MEAN.IO pour exécuter les différentes tâches nécessaires au développement, aux tests et à la mise en production est gulp. Il faut également installer bower pour la gestion des dépendances côté front-end, npm de Node.js étant utilisé côté back-end. Vous procéderez donc comme suit :

```
npm install -g gulp
npm install -g bower
```

Bien qu'il soit possible de cloner directement le dépôt GitHub de MEAN.IO, il est conseillé de passer par l'outil en ligne de commande dédié nommé mean-cli en l'installant via :

```
npm install -g mean-cli
```

Installation et configuration de MEAN.IO

Il est ensuite possible d'initialiser une application dans un dossier grâce aux commandes suivantes :

```
mean init folder_name
cd folder_name
npm install
```

MEAN.IO inclut un script (voir dossier *tools/scripts*) exécuté lors de la commande d'installation qui installe les dépendances back-end et front-end de tous les modules de l'application, il est donc inutile d'exécuter le classique `bower install`

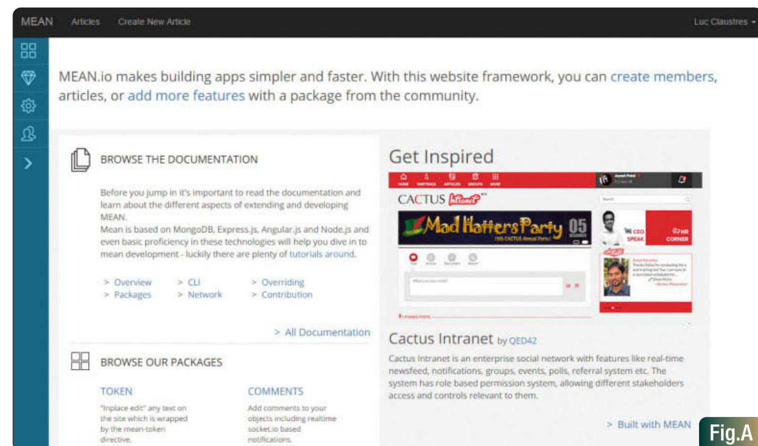


Fig.A

De la même façon il sera possible (nous y reviendrons plus tard) d'initialiser un package (i.e. un module) de l'application via :

```
mean package package_name
```

La première chose à faire est de créer une base de données avec un utilisateur ayant les droits d'accès, pour cela utiliser le shell de MongoDB en le lançant via votre compte administrateur de la base de données (configuré à l'installation) :

```
mongo admin --username root --password root
db = db.getSiblingDB('mean-dev')
db.createUser( { user: "mean-dev", pwd: "mean-dev", roles: [ "readWrite", "dbAdmin" ]} )
```

Il faut ensuite modifier la configuration par défaut de MEAN.IO pour utiliser cette base et cet utilisateur, pour cela ouvrir le fichier *development.js* dans le dossier *config/env* et changer la valeur de la clef **db** :

```
module.exports = {
 db: 'mongodb://mean-dev:mean-dev@localhost:27017/mean-dev',
 debug: true,
 ...
```

Pour lancer le serveur exécutez simplement la commande `gulp` puis connectez-vous avec votre browser à l'adresse http://localhost:3000/. Vous pouvez créer votre premier utilisateur pour vous connecter à l'application via l'entrée *Join* dans la barre de menu. A ce stade vous ne devriez avoir que deux entrées accessibles dans la barre de menu en plus du menu associé à votre profil (voir Figure 1). Pour rendre l'utilisateur créé administrateur de l'application utilisez la commande suivante :

```
mean user email@google.com --addRole admin
```

Une fois connecté une nouvelle barre de menu verticale apparaît sur la gauche et contient les différentes options de configuration accessibles en mode administrateur. **Fig.A.**

Trucs & Astuces : exécutez la commande suivante dans le dossier racine de l'application pour obtenir toutes les informations de version sur MEAN.IO (utile par exemple lors de la soumission de bugs sur le tracker) :

```
mean status
```

Mise à jour de MEAN.IO

Au final le dossier d'une application MEAN.IO est en fait un dépôt Git, puisque la commande `mean init` utilise Git pour installer le code du canevas. Elle créé par défaut un dépôt remote (distant) nommé **upstream**. Pour se maintenir à jour avec la dernière version il suffit donc de faire :

```
git pull upstream master
npm install
```

Maintenir à jour les prérequis peut parfois aider à corriger quelques bugs lors de l'installation :

```
npm update -g gulp
npm update -g bower
```

Fonctionnement

Le principal objectif de MEAN.IO est de fournir un canevas en termes d'usine logicielle et de structure de base, ainsi qu'un mécanisme d'extension, pour développer une application. Celui-ci passe par le développement d'un package ou d'un module qui est un ensemble de fonctionnalités (services back-end + composants front-end) permettant d'étendre le canevas. Comme il n'est pas forcément nécessaire pour une application donnée, il s'intègre au sein du canevas selon une règle bien définie si besoin tel un "plugin" (i.e. "greffon").

Environnements d'exécution

De façon classique une application MEAN.IO peut être lancée dans différents environnements :

- **development** : environnement utilisé pendant le développement ;
- **test** : environnement utilisé pour lancer les tests ;
- **production** : environnement utilisé pour la production.

Pour définir l'environnement il est possible de passer par la variable d'environnement **NODE_ENV** avant de lancer le serveur :

```
set NODE_ENV=production
gulp
```

Il est aussi possible d'invoquer directement le serveur en passant l'environnement en paramètre :

```
gulp production
```

Les tâches par défaut qui sont exécutées par gulp suivant l'environnement sont les suivantes :

- **development** : exécution de JSHint, CSSLint, Less, CoffeeScript sur le code et lancement du serveur en mode debug ;
- **test** : exécution de Karma et de Mocha ;
- **production** : minification du code CSS/JS des dépendances et lancement du serveur en mode production (multithread).

Le format de sortie des logs côté serveur dépend également de l'environnement, tiny en développement et combined en production :

```
// Environnement de développement
GET /system/views/index.html 304 2.379 ms - -
// Environnement de production
::1 - - [22/Mar/2015:13:13:42 +0000] "GET / HTTP/1.1" 200 - "-" "Mozilla/5.0 (Macintosh;
 Intel Mac OS X 10_10_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.
101 Safari/537.36"
```

Anatomie d'une application

Une fois initialisé le dossier d'une application MEAN.IO présente la structure suivante :

```
Application folder
--- bower_components : contient les dépendances Bower (front-end)
--- config : fichiers de configuration (Express, base de données, etc.)
  --- env
  --- middlewares
--- gulp : fichiers décrivant les différentes tâches gulp en fonction de l'environnement
--- logs : contient les logs en sortie de Node.js
--- node_modules : contient les dépendances Node.js (back-end)
--- packages : contient l'ensemble des modules de l'application
  --- core : modules de base inclus avec MEAN.IO
  --- custom : modules spécifiques à la logique applicative
--- tools :
  --- scripts
  --- test
```

A la racine, on trouve tous les fichiers de configuration pour les outils de l'usine logicielle que sont npm, bower, gulp, jshint, karma, protractor, etc. Le répertoire **config** contient l'ensemble des fichiers de configuration, notamment *express.js* et le dossier **middlewares** (logging, etc.) pour Express et le dossier **env** pour les options propres à MEAN.IO.

Les options communes à tous les environnements sont stockées dans le fichier **env/all.js**, les options propres à chaque environnement sont stockées dans un fichier portant le nom de l'environnement dans le dossier **env**. Une liste non exhaustive des options de configuration est la suivante :

- **root** : le chemin vers la racine de l'application ;
- **db** : l'URL d'accès à la base de données, peut inclure un login/password de la forme *mongodb://login:password@host:port/base ;*
- **hostname** : le nom de l'hôte ;
- **http/https.port** : le numéro de port à utiliser via HTTP ou HTTPS ;
- **logging.format** : format morgan des logs serveur ;
- **app.name** : le nom de l'application ;
- **aggregate** : true/false pour activer/désactiver l'agrégation (utile en mode debug) ;
- **secret** : clef privée pour la sécurisation via JWT (voir ci-après) ;
- Informations pour se connecter via des réseaux sociaux.

Anatomie d'un module

Le dossier d'un package ou module MEAN.IO présente la structure suivante :

```
Module folder
--- docs : contient la documentation de l'API (back-end)
--- public : partie publique du module sur le site
  --- assets : données statiques (images, css, dépendances front-end)
  --- controllers : controllers front-end (AngularJS)
  --- routes : routing front-end (AngularUI Router)
  --- services : services front-end (AngularJS)
  --- tests : tests front-end (Jasmine)
  --- views : vues (AngularJS)
--- server : contient le code back-end
  --- controllers : controllers back-end (Express)
  --- routes : routing back-end (Express)
  --- models : modèle de données back-end (Mongoose)
  --- tests : tests back-end (Jasmine)
  --- views : vues HMTL (templating Swig)
--- node_modules : contient les dépendances Node.js du module (back-end)
```

A la racine on trouve comme dans le cas de l'application les fichiers de configuration pour npm, bower et MEAN.IO (**mean.json**). Le plus important est le fichier **app.js** qui est le point d'entrée du module. Tous les fichiers à

l'intérieur du dossier **public** seront accessibles publiquement à l'URL */nom-module/chemin-relatif-fichier*. Par exemple pour accéder à un contrôleur AngularJS nommé 'controller' dans le module nommé 'module' l'URL sera *module/controllers/controller.js*.
Pour rajouter un module au canevas il suffit de lui donner un nom unique dans l'application et de le copier dans le répertoire **packages/custom** de l'application; MEAN.IO se charge du reste !

Authentification

Le canevas inclut la gestion des utilisateurs et de leurs rôles. Vous disposez donc d'une page pour enregistrer un nouvel utilisateur, d'une page pour se connecter à l'application, d'une page de réinitialisation du mot de passe (nécessite toutefois de configurer un serveur SMTP), et d'une page d'index accessible de façon publique. Si vous avez le rôle administrateur ('admin') vous héritez aussi d'une IHM de gestion des utilisateurs (ajout, suppression, affectation de rôle).
Si l'authentification permet d'adapter le contenu de l'application (menus et pages) en fonction du rôle de l'utilisateur, elle sert également à protéger l'accès à l'API côté back-end. Pour ce faire elle se base sur JSON Web Token (JWT), qui est une spécification pour l'authentification. Un JWT est un objet JSON que le serveur encode en utilisant une clé privée. L'objet JSON encodé (qui est en fait l'utilisateur dans MEAN.IO) se présente sous la forme d'un token renvoyé au client qui s'est authentifié avec succès. Dans MEAN.IO cela se passe lors de la connexion qui déclenche un appel vers l'URL*/api/login* de l'API. Ensuite, grâce à un intercepteur AngularJS, la partie cliente MEAN.IO associera ce token à chaque requête faite au serveur. Si en utilisant sa clé privée le serveur parvient à décoder le token, il s'assure de l'authenticité du client et autorise la requête.

Agrégation

Afin de pouvoir rajouter un module au canevas sans devoir modifier un quelconque fichier, MEAN.IO intègre un mécanisme automatique d'agrégation, tant pour les fichiers applicatifs que pour les dépendances. Celui-ci repose principalement sur du templating côté serveur, par défaut tous les fichiers JS du dossier **public** de chaque module sont agrégés à l'exception du sous dossier **assets** réservé pour le stockage des librairies externes, des images et des fichiers CSS. En environnement de production les fichiers sont également minifiés.
Concernant les dépendances MEAN.IO offre deux possibilités :

- L'utilisation du fichier global **config/assets.json** qui contient une liste de fichiers JS/CSS externes ;
- L'ajout des fichiers externes directement via le code source de chaque module.

La première approche est utilisée de façon interne par MEAN.IO pour les dépendances globales du canevas comme AngularJS ou encore jQuery. Rien ne vous empêche de faire de même pour votre application, néanmoins ceci la rendra plus monolithique dans le sens où les dépendances de vos différents modules seront stockées de façon globale. Il deviendra donc impossible d'ajouter ou de supprimer un module simplement en déplaçant son dossier. L'avantage est par contre d'avoir toutes les dépendances localisées à un seul endroit : **bower_components** pour Bower (front-end) et**node_modules** pour Node.js (back-end). La seconde approche permet à chaque module de déclarer ses dépendances et donc de rester complètement indépendant du canevas. Le chemin des fichiers JS et CSS doit être donné relativement aux dossiers **public/assets/js**et **public/assets/css**. Ceci se fait dans le fichier **app.js** du module :

```
// Ajout d'une librairie externe
Module.aggregateAsset('js','lib.js');
```

```
// Ajout du fichier CSS du module
Module.aggregateAsset('css','module.css');
```

Il est possible de contrôler l'agrégation plus finement si nécessaire :

```
// Ajout d'une librairie externe dans le header et avant les suivantes (par défaut le poids vaut 0)
Module.aggregateAsset('js','first.js',{weight: -1, group: 'header'});
// Ajout d'une librairie externe dans le footer
Module.aggregateAsset('js','last.js',{group: 'footer'});
```

Avec cette approche les dépendances d'un module sont généralement installées localement au module. Le script d'installation de MEAN.IO parcourt en effet automatiquement tous les modules et exécute à l'intérieur un `npm/bower install`. NPM installera donc les dépendances indiquées du fichier **package.json** dans le dossier **node_modules** et il semble impossible de changer ce comportement par défaut. Néanmoins, comme NPM utilise une stratégie récursive de recherche des modules cela ne pose aucun problème particulier. La stratégie concernant Bower consiste, via un fichier**.bowerrc**, à lui faire installer les dépendances du fichier **bower.json** dans le dossier **public/assets/js**.

```
{
  "directory": "public/assets/lib"
}
```

Contribuer

La société qui a lancé MEAN.IO (Linnovate) a également déployé une infrastructure afin de permettre à la communauté de partager et de mettre à disposition des modules, il s'agit du MEANetwork. La première chose pour utiliser cette infrastructure est de s'enregistrer via l'outil en ligne de commande :

```
mean register
```

Une fois enregistré vous pouvez publier un module en vous positionnant dans le dossier du module et en exécutant la commande `publish` :

```
cd packages/custom/module
mean publish
```

Conclusion

Nous avons passé en revue dans cet article les grands principes de MEAN.IO. Pour ma part, même s'il est évident que le framework n'est pas encore mature (la version 0.5 courante au moment de la rédaction de l'article est annoncée comme la release candidate à la version 1.0), il me semble déjà procurer une bonne base pour structurer des applications Web SPA full JavaScript :

- Une architecture modulaire orientée composant (un module est un plugin qui peut être rajouté à l'application sans nécessité de modifier l'existant) ;
- Une organisation de fichiers à l'intérieur de chaque module qui permet de simplifier le développement en automatisant la déclaration des modèles, des routes et des dépendances ;
- Une usine logicielle (analyse de code, minification, documentation, etc.) ;
- Des fonctionnalités de base requises par la plupart des applications (authentification, gestion des rôles, gestion des menus, options de configuration, etc.) ;
- Un réseau naissant de modules d'extension portés par la communauté ;

Dans les prochains articles nous aborderons la création et le fonctionnement détaillé d'un module MEAN.IO avec un cas d'utilisation concret.



Unity 5.1 : créer son jeu multijoueur avec la nouvelle API

La sortie de Unity 5.1 a apporté son lot de nouveautés et parmi elles, un tout nouveau système pour créer des jeux multi-joueurs : UNET. Plus souple et accessible que son prédécesseur, il propose une surcouche d'abstraction à toutes les problématiques réseaux afin que le développeur n'ayant aucune ou peu de compétence en réseau puisse facilement créer son jeu multi. Le but avoué d'Unity étant que tout le monde soit en mesure de créer le MMO de demain !



Teddy DESMAS - Daniel DJORDJEVIC
Maxime FRAPPAT
Consultants Infinite Square



Le système se compose de deux parties en fonction des connaissances en réseau et du type de jeu :

- Pour les utilisateurs novices qui souhaitent simplement faire un jeu multi-joueurs aisément, les équipes de développement ont fourni la « High Level API »
- Pour les plus barbus d'entre nous, qui veulent construire toute une infrastructure réseau, optimiser les paquets réseaux et plus encore, il faudra se tourner vers la « NetworkTransport API »

High Level API

Derrière le nom de code « HLAPI » se cache un moyen d'accéder aux fonctionnalités les plus souvent utilisées lors de la création d'un jeu multijoueur en oubliant toute la partie *low level* qui sera gérée toute seule. Voici ce à quoi cette API vous donne accès :

- Contrôler l'état du réseau du jeu grâce à la classe « Network Manager » ;
- Créer une partie pouvant accueillir d'autres joueurs, le joueur hébergeant lui-même la partie ;
- Envoyer et recevoir des messages ;
- Envoyer des commandes des clients vers le serveur ;
- Passer des commandes RPC (Remote Procedure Call) à partir des serveurs aux clients ;
- Envoyer des évènements des serveurs aux clients.

UNET Services

En parallèle de l'API, Unity propose un service permettant de gérer votre jeu en production. Ceci inclut un serveur multijoueur pour permettre à votre jeu de communiquer à travers Internet sans avoir besoin d'une adresse IP publique. Le trafic réseau passe par un serveur relais hébergé par Unity dans le Cloud au lieu de passer directement entre les clients. Ceci évite les problèmes avec les pare-feu et NATS, permettant aux joueurs de se joindre depuis n'importe où.

Le service fournit aussi la capacité pour les utilisateurs de créer des matchs et de faire de la publicité pour ses matchs, afficher les matchs disponibles pour les rejoindre.

Pour gérer tout cela dans votre jeu, il suffit d'ajouter le composant « NetworkMatch » et d'associer un script comme celui-ci, permettant de créer, rejoindre et quitter une partie :

```
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.Networking.Types;
using UnityEngine.Networking.Match;
using System.Collections.Generic;
```

```
public class HostGame : MonoBehaviour
{
  List<MatchDesc> matchList = new List<MatchDesc>();
  bool matchCreated;
  NetworkMatch networkMatch;

  void Awake()
  {
    networkMatch = gameObject.AddComponent<NetworkMatch>();
  }

  void OnGUI()
  {
    if(GUILayout.Button("Create Room"))
    {
      CreateMatchRequest create = new CreateMatchRequest();
      create.name = "NewRoom";
      create.size = 4;
      create.advertise = true;
      create.password = "";

      networkMatch.CreateMatch(create, OnMatchCreate);
    }

    if (GUILayout.Button("List rooms"))
    {
      networkMatch.ListMatches(0, 20, "", OnMatchList);
    }

    if (matchList.Count > 0)
    {
      GUILayout.Label("Current rooms");
    }

    foreach (var match in matchList)
    {
      if (GUILayout.Button(match.name))
      {
        networkMatch.JoinMatch(match.networkId, "", OnMatchJoined);
      }
    }
  }

  public void OnMatchCreate(CreateMatchResponse matchResponse)
  {
    if (matchResponse.success)
```

```
    {
      Debug.Log("Create match succeeded");
      matchCreated = true;
      Utility.SetAccessTokenForNetwork(matchResponse.networkId, new NetworkAccess
Token(matchResponse.accessTokenString));
      NetworkServer.Listen(new MatchInfo(matchResponse), 9000);
    }
    else
    {
      Debug.LogError ("Create match failed");
    }
  }

  public void OnMatchList(ListMatchResponse matchListResponse)
  {
    if (matchListResponse.success && matchListResponse.matches != null)
    {
      networkMatch.JoinMatch(matchListResponse.matches[0].networkId, "", OnMatch
Joined);
    }
  }

  public void OnMatchJoined(JoinMatchResponse matchJoin)
  {
    if (matchJoin.success)
    {
      Debug.Log("Join match succeeded");
      if (matchCreated)
      {
        Debug.LogWarning("Match already set up, aborting...");
        return;
      }
      Utility.SetAccessTokenForNetwork(matchJoin.networkId, new NetworkAccessToken
(matchJoin.accessTokenString));
      NetworkClient myClient = new NetworkClient();
      myClient.RegisterHandler(MsgType.Connect, OnConnected);
      myClient.Connect(new MatchInfo(matchJoin));
    }
    else
    {
      Debug.LogError("Join match failed");
    }
  }

  public void OnConnected(NetworkMessage msg)
  {
      Debug.Log("Connected!");
    }
  }
```

Intégration dans l'éditeur

Toute la couche networking fait partie intégrante du moteur et de l'éditeur ce qui permet d'avoir accès à des composants et des aides visuelles afin de vous aider à créer votre jeu.
On retrouve par exemple, le composant « NetworkIdentity » qui permet de lier un objet au réseau. Il est utilisé pour synchroniser les informations de l'objet via le réseau. Concrètement dans le cas d'un FPS, si vous changez la position de votre personnage l'information sera automatiquement envoyée au serveur. C'est au serveur que revient la tâche de créer les instances des objets ayant une « NetworkIdentity » autrement la liaison ne se fera pas correctement avec le système.
La classe « NetworkBehaviour » sert quant à elle lors de la création de script voulant utiliser les commandes RPC. De nombreux évènements sont aussi accessibles comme « OnStartClient() » qui permet d'être notifié quand le client se connecte au serveur. On l'utilise en remplaçant simplement l'héritage de « MonoBehaviour » vers « NetworkBehaviour ».
Bien entendu, le placement des objets dans la scène est supporté nativement, ce qui permet de ne pas perdre du temps à gérer cela manuellement.

Créer son premier jeu multijoueur

Mise en place

Rien de mieux qu'un peu de pratique pour vérifier par soi-même si, effectivement, la promesse de Unity est tenue. Pour ce tutoriel, on partira du postulat que vous maitrisez les bases pour créer un jeu simple en 3D, on ne s'attardera pas sur les actions de base (création de scènes et d'objets, ajout de composants, ...). C'est parti pour la création d'un nouveau projet 3D ! Notre but va être de rejoindre une partie et de pouvoir nous déplacer dans la scène. Pour faciliter la mise en place, nous allons utiliser un prefab du package « Characters » donc n'oubliez pas de l'ajouter via le menu *Assets > Import Package > Characters*. Ajoutez à la scène un plan qui nous servira de sol puis rajoutez deux cubes de différentes tailles posés sur le sol qui serviront d'obstacles.

Création du personnage

Comme dit précédemment, nous allons utiliser le prefab se trouvant dans le dossier *Assets\Standard Assets\Characters\FirstPersonCharacter\Prefabs\FPSController* et l'ajouter dans la scène. Renommez-le en *Player*, réinitialisez ses propriétés et placez-le correctement dans la scène. Dans le *Player*, ajoutez un nouvel objet de type « Capsule » puis supprimez son composant « Collider ». A ce stade, si vous testez le jeu, vous devez être en mesure de vous déplacer dans la scène.

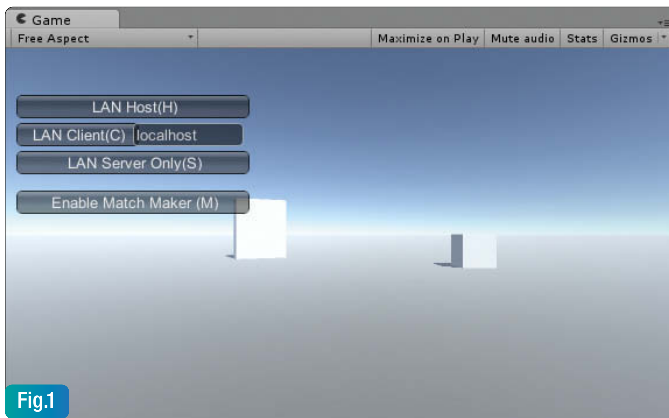


Fig.1

Paramétrage et synchronisation du personnage

Afin de rendre le personnage utilisable par le système, nous allons devoir lui rajouter le composant « NetworkIdentity ». Rajoutez le composant et cochez l'option *Local Player Authority.*

Un autre composant va nous être utile, « NetworkTransform », car il va permettre de synchroniser la position du joueur. Réglez la propriété *Transform Sync Mode* sur *Sync Transfrom*.

Pour ne contrôler que son joueur, nous allons devoir créer un script spécifique qui n'autorise pas les autres clients à contrôler les autres personnages. Le script se présente de cette façon :

```
using UnityEngine;
using UnityEngine.Networking;
using UnityStandardAssets.Characters.FirstPerson;

public class NetworkPlayerBase : NetworkBehaviour
{
   void Start()
   {
      if (isLocalPlayer)
      {
         GameObject.Find("Main Camera").SetActive(false);
         GetComponent<CharacterController>().enabled = true;
         GetComponent<FirstPersonController>().enabled = true;

         var fpsCharacter = transform.FindChild("FirstPersonCharacter");
         fpsCharacter.GetComponent<AudioListener>().enabled = true;
         fpsCharacter.GetComponent<Camera>().enabled = true;
      }
   }
}
```

Désactivez sur le *Player* les composants « Character Controller » et « First Person Controller », dans son objet fils *FirstPersonCharacter*, désactivez la « Camera » et l' « Audio Listener ». Tous ces éléments seront réactivés dans le script uniquement pour notre personnage.

Enregistrez le *Player* en tant que prefab afin de le réutiliser plus tard. Fig.1

NetworkManager

Pour la mise en place de la partie multi-joueurs, nous allons avoir besoin d'un nouvel objet vide que l'on nommera NetworkManager auquel on rajoutera deux composants « NetworkManager » et « NetworkManager HUD ». Dans le premier composant, on remarque dans la section *Spawn Info* que l'on peut renseigner le *Player Prefab*.

Pour pouvoir lier notre *Player* au composant, il doit obligatoirement posséder un composant de type « NetworkIdentity ». Faites la liaison entre le

Interview de John Riccitiello lors de la conférence Unite Europe

Bilan : moins d'un an après votre nomination en tant que CEO chez Unity Technologies, quel est votre premier bilan ?

Il y a des choses qui ne changent pas, nous restons fidèles aux mêmes principes : la démocratisation, la résolution de problèmes difficiles, ou encore aider les gens à obtenir plus de succès pour leur entreprise. Certaines choses sont différentes : nous mettons davantage l'accent sur certains domaines, comme aider les gens à faire de beaux jeux. Nous savons que les développeurs nous poussent à le faire, donc nous avons beaucoup investi et beaucoup embauché pour explorer les façons dont nous pouvons créer de plus beaux jeux. Nous voulons accélérer l'innovation, nous investissons pour être un partenaire plus fort.

Après le succès de Hearthstone : Heroes of Warcraft développé par Blizzard sur Unity, comptez-vous cibler de façon plus systématique les grands éditeurs comme Activision, Ubisoft, EA... ? Est-ce dans la stratégie à long terme d'Unity de conquérir ces nouveaux marchés ?

Activision, EA et beaucoup d'autres sont déjà nos clients, nous travaillons avec eux. Ce serait une mauvaise stratégie de se concentrer sur un type particulier d'utilisateur. Nous avons des studios composés de seulement 9 personnes par exemple qui veulent concurrencer les jeux d'EA. Donc nos outils sont là pour aider les petites mais aussi les grandes entreprises à faire des jeux plus beaux et tout le monde les apprécient.

Vous avez annoncé un partenariat avec Microsoft en ce qui concerne le développement sur Hololens. Comment imaginez-vous le futur du jeu vidéo grâce à la réalité augmentée ?

La réalité virtuelle va exploser mais pas immédiatement. Cela va prendre du temps. Nous n'avons pas encore trouvé le moyen de proposer une véritable expérience avec ces outils pour le moment mais nous y travaillons.

Quel est votre jeu coup de cœur développé sur Unity ? Pourquoi ?

Le mois dernier j'ai beaucoup joué à Empires and Allies et ensuite j'ai arrêté pour rejouer à Bioshock Infinite pour montrer à ma fille que ce jeu est magnifique. J'ai joué à Battlefiled Hardline pendant un moment, je l'aime bien. J'attends avec impatience le nouveau jeu Star Wars qui arrive prochainement. Chaque fois que je m'ennuie je fais quelques niveaux à Candy Crush, je ne joue plus aux jeux de tir car je ne suis plus assez rapide. J'aime jouer aux FPS mais le multijoueur est trop compétitif pour moi. Parfois je joue avec des joueurs meilleurs que moi en équipe mais je me fais tuer à chaque fois. Il me donne toujours le rôle de tank pour que je survive un peu plus longtemps.

Comment se passent les conférences Unite à travers le monde ? Sentez-vous des évolutions différentes dans la production de jeux selon les continents ?

Il y a un gros malentendu sur les jeux de nos jours, les gens pensent qu'il y a des marchés séparés pour les consoles et le mobile. C'est naïf. On ne devrait pas segmenter sur le type de plateforme mais sur la densité de contenu de chaque jeu. Il y a la gamme supérieure, les jeux profonds et riches en contenu et qui peuvent se jouer autour de 40/50 heures comme Assassin's Creed, GTA, Fifa, COD, Bioshok... Les jeux de moyenne densité : moins riches, moins d'heure pour les finir comme World of Tanks. Les jeux light comme Candy Crush. C'est pour ça que les consoles ont de l'avenir. Ce n'est pas pour le côté hardware, c'est pour les jeux proposés sur ces plateformes. Je n'ai jamais vu un jeu à grande densité sur mobile. Par exemple, tu ne vas pas t'arrêter de jouer à FIFA pour jouer à Candy Crush, même si tu joues aux deux d'habitude.

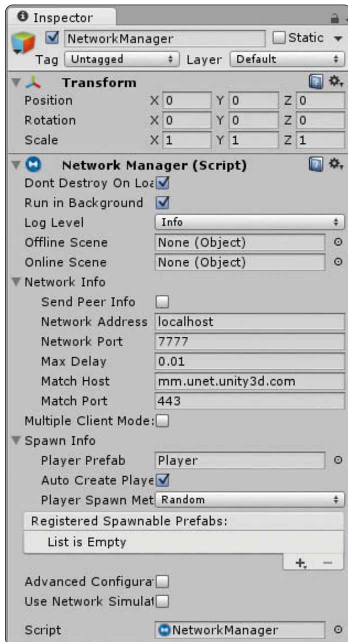


Fig.2

prefab du joueur et la propriété *Spawn Info*. Fig.2.

Et voilà ! Vous pouvez tester le jeu en lancer deux versions en même temps pour apprécier le résultat : vous venez de créer votre premier jeu multijoueur J

ROADMAP : OÙ VA UNITY ?

Lors de l'Unite 2015, la conférence annuelle d'Unity Technologies, les équipes ont bien entendu annoncé de nombreuses nouveautés techniques, mais aussi dévoilé leur roadmap interne. Accessible à l'adresse https://unity3d.com/unity/roadmap, on découvre ce qui attend les utilisateurs du célèbre éditeur Unity pour les prochains mois ainsi que les plans pour les années futures.

Les 3 prochaines versions

Version 5.2

La prochaine version sortie le 8 septembre 2015 prévoit un bon nombre de fonctionnalités, dont les plus importantes :

- Audio : Spatialization SDK
- Core : Des options pour la StackTrace dans les logs
- Graphismes : Optimisations CPU, Deferred Reflection Probes
- Networking : WebRequest (experimental)
- Platform : Support d'IL2CPP pour Android, PS4 et Xbox One (experimental)
- Platform : Support des Windows 10 Universal Apps
- Services : Intégration d'Unity Ads et d'Unity Analytics
- UI : Nouveaux contrôles comme Dropdown, 2D Clipping etc.

Version 5.3

La version 5.3 sortira le 8 décembre 2015 et rendra heureux les nombreux utilisateurs de MonoDevelop avec le support de la version 5.9 de l'IDE. :

- 2D : Nouveaux type de jointure (Joints)
- Editeur : Edition multiple de scènes, support du Retina/HDPI, zoom dans la vue « Game »
- Graphics : Support de DirectX12 et d'OpenGL 4.x, debogueur de frame à distance
- Réseau : Support des flux JSON
- Platform : support d'IL2CPP pour les Windows Store Apps

Version 5.4

Pour la version 5.4 qui sortira le 16 mars 2016, Unity n'a livré que peu d'informations sur son contenu réel mais on sait d'ores et déjà qu'il y aura au moins :

- IA : NavMesh workflow
- Déploiement : Reproductibilité binaire
- MonoDevelop : Intégration d'Unity REST
- Réseau: Serveur de simulation autonome

Les évolutions en développement

Les équipes de développement travaillent également sur les futurs sujets qui ont été approuvés. Ces nouveautés apparaîtront dans les prochaines versions mais aucune date n'est encore disponible à ce stade :

- Editeur : Support de Linux
- Editor : Sauvegarde des modifications en mode « Play »
- Platform : Support de l'Apple Watch, iOS 9, New Nintendo 3DS, Microsoft HoloLens, OS X Metal support, WebGL 2.0 et XCode 7

Les évolutions en recherche

Bien entendu, Unity continue d'explorer des pistes de développement qui sont souvent des sujets à long terme ou sensibles. En voici des exemples :

- 2D : Animation des personnages
- Animation : Animation dépendent de la physique
- Mise à jour du profil .NET Profile
- Scripting visuel

Conclusion

Avec l'ouverture de la roadmap au public, Unity joue clairement la carte de la transparence et tente de rassurer les développeurs qui commençaient à s'impatienter sur plusieurs sujets critiques et réclamés de longue date. Espérons qu'ils arrivent à tenir leurs promesses car l'attente est grande !



TECHDAYS CAMPS

Microsoft organise une série de conférences dans toutes les France. Ces journées s'organisent autour de 4 thèmes : Windows 10, développement web, Données et IT, avec notamment Azure, Office 365, Windows Server. Les dates à venir :

- **Paris : 15 octobre, 26 novembre**
- **Toulouse : 6 octobre**
- **Strasbourg : 5 novembre**
- **Lyon : 1er octobre**
- **Nantes : 8 octobre**

Pour en savoir et inscription :
https://techdays.microsoft.fr/camp/default.aspx#pg1

octobre

Kaiz'n day
08 octobre à 14h (Zenika Paris)

Zenika lance sa 1ère édition de Kaiz'n day : baladez-vous entre gaming et actualités sur les nouvelles technos ! Nous proposons des ateliers et conférences sur : Docker, Angular 2, l'Internet des Objets (IoT) et l'Agilité. Programme et inscription :
https://jobs.zenika.com/kaiznday/

Matinale projet Agile externalisé :
retour d'expérience client / 09 octobre (Zenika Lyon)

Au cours de cette matinale, vous découvrirez les fondamentaux de la démarche mise en place. Nous décrypterons ensuite les pratiques agiles en situation d'externalisation.
Enfin Real.not vous donnera son retour d'expérience concret. Programme et inscription : http://www.zenika.com/matinale-zenika-lyon-real-not-zfactory.html

Lancement officiel Visual Studio Studio 2015
Issy-les-Moulineaux : 15 octobre

L'édition 2015 offre de nouvelles capacités, permettant à tous les développeurs d'augmenter leur productivité et leurs possibilités de développement sur de multiples plateformes, de Windows à Linux en passant par iOS et Android...
Site : http://www.microsoft.com/france/visual-studio/evenements/default.aspx?current=false

NetBeans Day Paris : 16 octobre

La communauté NetBeans organise un événement dédié autour de son IDE, relativement peu connu en France. La conférence sera l'occasion de rencontrer les utilisations NetBeans, de découvrir l'environnement si vous ne connaissez pas et plusieurs pointures du monde Java/NetBeans seront là.
L'agenda est copieux : pourquoi NetBeans, développement Java EE avec Maven et WildFly, Node.js, créer des plug-ins, Angular, bien démarrer avec NetBeans, trucs et astuces avancés.
Lien : https://www.eventbrite.com/e/netbeans-day-paris-tickets-18423155153?aff=erelexpcat

UCAAT 3e édition (Sophia Antipolis)
20-22 octobre

Cette conférence se dédie à l'automatisation avancée des tests. Durant ces 3 journées, il sera question des récentes nouveautés, d'outils, de méthodes, les problématiques des tests. De nombreux thèmes seront abordés : tests chez Unity, TDL, tests dans le mobile, les modèles.
Pour en savoir plus : Plus d'information sur :
http://ucaat.etsi.org/2015/index.html ou par mail à l'adresse suivante UCAAT@etsi.org

novembre

La Droidcon est de retour sur Paris les 9 et 10 novembre au Tapis Rouge !

Les fans d'Android seront ravis de cette nouvelle : LA plus grande messe des développeurs 100% Android revient bientôt et promet d'être encore plus intense que les années précédentes. Co-organisée par le Paris Android User Group (PAUG) et BeMyApp, la Droidcon à Paris se prépare déjà et vous attend nombreux pour sa troisième édition. L'événement porte sur trois thèmes principaux : Android Development, Android Everywhere et UX/UI. Entrée payante.

Forum PHP : 23 & 24 novembre (près de Paris)

Le monde PHP fête les 20 ans du langage et de la sortie de 7 PHP. Le Forum PHP est la référence française des développeurs et entreprises utilisant ce langage open source ! Un des thèmes abordés sera 7 PHP qui sortira presque en même temps. Une track dédiée sera proposée et le sujet promet d'être très largement abordé ! Site : http://www.afup.org/pages/forumphp2015/

DevFest 2015 à Nantes : 6 novembre

Pour cette 4eme édition, l'accent sera mis sur les nouvelles technologies et l'innovation : il sera question des dernières avancées en terme de Cloud, Web, Mobile & IoT. Des experts locaux et internationaux seront présents :

- Francesc Campoy Flores, équipe Go chez Google
- Raphaël Goetter, expert web et créateur d'Alsacréations
- Cyril Mottier, Google Developer Expert Android chez Capitaine train
- Goeffrey Dorne, Expert UX chez Human & Design

Ce DevFest est le rassemblement français autour des technologies Google avec plus de 700 participants. Cet évènement soutenu par Google, Intel... et plus d'une vingtaine de partenaires locaux est labelisé NantesTech.
Site Web : http://devfest.gdgnantes.com

Evénement : OW2con'15 - 17 & 18 novembre

La septième édition de la conférence annuelle de la communauté open source OW2 aura lieu les 17 et 18 novembre 2015, à Paris sur deux sites distincts. Cet évènement rassemble des experts informatiques, architectes, développeurs et responsables de projets, en provenance du monde entier :
Site : http://www.ow2con.org/

LabVIEW Developer Days (National Instrument),
du 17 au 26 novembre dans 8 villes de France.

Cette journée est dédiée aux programmeurs et aux développeurs LabVIEW. Objectif : renforcer vos compétences techniques et vos connaissances de la plate-forme NI au travers de nombreuses sessions techniques : des principes de conception logicielle pour créer du code modulaire aux modèles de conception, en passant par les bonnes pratiques de programmation qui vous assureront la stabilité et la maintenabilité de vos applications à long terme. Vous pourrez également passer gratuitement la certification CLAD – Certified Associate Developer.

- **17 novembre - Paris**
- **19 novembre - Rouen**
- **24 novembre - Marseille**
- **26 novembre - Bordeaux**
- **17 novembre - Rennes**
- **19 novembre - Strasbourg**
- **24 novembre - Toulouse**
- **26 novembre - Grenoble**

Inscription gratuite et infos sur http://france.ni.com/labview-developer-days

Open For Innovation Paris Open Source Summit

Le 1er événement européen du Libre et de l'Open Source se tiendra les 18 et 19 novembre 2015 aux Docks de Paris.
Au programme 2015 :·90 conférences, 150 exposants et partenaires, Le village associatif, La student demo Cup, L'Open CIO Summit, Le Lab Open Source, Le village innovation. Retrouvez tout le programme et demandez votre badge d'accès sur www.opensourcesummit.paris

GEEKOFYOU : l'impression 3D accessible à tous



FAHRASMANE Mike
Titulaire d'un master en génie électrique et acteur du mouvement maker et DIY (Do-It-Yourself) passionné d'innovation. Il est le Fondateur de GEEKOFYOU
www.geekofyou.fr
contact@gekofyou.fr



A l'heure où nous parlons, l'impression 3D est en train de tracer sa route. Très bien j'ajouterai, car elle est bien aidée par les entreprises de développement qui ne cessent de concevoir, développer et innover dans le secteur. Malheureusement, certains particuliers, salariés et entreprises n'ont pas encore franchi le pas pour en acquérir une ou en utiliser une par peur de l'inconnu.

Les nouvelles technologies m'ont toujours fasciné depuis mon jeune âge et cela a été confirmé par le choix de mes études (Bac STI électronique pour finir par un master professionnel en génie électrique et informatique industrielle).

Durant mes différentes expériences professionnelles, j'ai été amené à travailler au sein de bureaux d'études en électronique ou il était question de concevoir des prototypes pour un cahier des charges bien précis. Mon équipe devant concevoir la carlingue du système, c'est à ce moment qu'est apparue l'impression 3D sur ma route, il y a quelques années maintenant.

Geekofyou a été fondée dans le but de partager le savoir de l'impression 3D en proposant diverses formations permettant de s'approprier l'impression 3D et son environnement.

Il faut savoir que l'impression 3D de nos jours touche à plusieurs secteurs:

- L'ingénierie ;
- La médecine ;
- L'aérospatiale ;
- L'agroalimentaire ;
- La mode ;
- La joaillerie et bien plus encore.

C'est pour cela que nous proposons différents stages d'une durée de plusieurs jours qui vont de l'utilisation d'une imprimante 3D, de scanner 3D, de la construction d'une imprimante 3D Reprap et de l'usage de tous les logiciels nécessaires au paramétrage de notre impression.

L'objectif à la fin du stage est de pouvoir :

- Utiliser une imprimante 3D de façon autonome et en toute sécurité ;
- Optimiser ses impressions ;
- Utiliser les différents logiciels de l'environnement 3D ;
- Dépanner son imprimante ;
- Savoir dessiner une pièce avec un logiciel de CAO ;
- Connaître les différentes parties de l'imprimante 3D ;
- Utiliser les différentes méthodes de scan 3D ;

Les différents stages s'adressent à tous publics. On parle là du salarié, du professionnel, des entreprises, des particuliers en passant par les passionnés.

L'objectif étant de satisfaire tout le monde, nous proposons chaque mois des stages sur Paris. Nous nous déplaçons aussi vers les entreprises qui souhaitent proposer les formations dans leurs locaux à plusieurs de leurs salariés, et ça dans toute la France.

Afin d'assurer le bon déroulement des stages, nous mélangeons théorie et cas pratiques grâce à nos imprimantes pour les exemples concrets.

L'impression 3D étant en constante évolution, il est important d'avoir de bonnes bases afin de bien maîtriser toutes les parties nécessaires pour le bon déroulement de vos impressions. Gardez en tête que cette technologie s'insinue de plus en plus dans tous les domaines. Vous serez amenés à croiser ou à en entendre parler demain ou après-demain sur votre route.

Si comme tous les passionnés vous souhaitez plus d'informations ou assister à un stage, vous le pouvez en cliquant sur le lien : geekofyou.fr/formation/ ou par mail : contact@geekofyou.fr



Abonnement : Service Abonnements PROGRAMMEZ, 4 Rue de Mouchy , 60438 Noailles Cedex. - Tél. : 01 55 56 70 55 - *abonnements.programmez @groupe-gli.com* - Fax : 01 55 56 70 91 - du lundi au jeudi de 9h30 à 12h30 et de 13h30 à 17h00, le vendredi de 9h00 à 12h00 et de 14h00 à 16h30. **Tarifs** abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € Autres pays : nous consulter.
PDF : 30 € sur www.programmez.com



Une publication **Nefer-IT**
7 avenue Roger Chambonnet
91220 Brétigny sur Orge
redaction@programmez.com
Tél. : 01 60 85 39 96

Directeur de la publication & rédacteur en chef : François Tonic

Secrétaire de rédaction : Olivier Pavie

Ont collaboré à ce numéro : S. Saurel

Experts : N. Freier, O. Pavie, C. Michel, C. Pichaud, J. Corioland, W. Chegham, A. Alnijres, L. Pierron, B. Lacherez, Y . Benoit, M. Fahrasmane, T. Desmas, D. Djordjevic, M. Frappat, L. Claustres, P. Lemberger, J. Valloire, F. Marco, M. Kuznetsov, L. Vaylet.

Photos/illustrations : © ÉDITIONS SOLEIL/GANG/LABOUROT

Publicité : PC Presse,
Tél.: 01 74 70 16 30, Fax : 01 41 38 29 75
pub@programmez.com

Maquette : Pierre Sandré

Imprimeur : S.A. Corelio Nevada Printing, 30 allée de la recherche, 1070 Bruxelles, Belgique.

Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude

Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD
Téléphone : 09 67 32 09 34

Contacts
Rédacteur en chef : ftonic@programmez.com
Rédaction : redaction@programmez.com
Webmaster : webmaster@programmez.com
Publicité : pub@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1215 K 78366 - ISSN : 1627-0908